UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN PERSONAJE SINTÉTICO INTELIGENTE PARA UN VIDEOJUEGO DE ACCIÓN EN PRIMERA PERSONA.

Autor: Francisco Jesús González López

Tutor: Raúl Arrabales Moreno

Julio 2009

Agradecimientos

A mis padres y a Ana, por su cariño, su santa paciencia y por haber estado siempre a mi lado; y a mis hermanas Marta y Amelia, porque este triunfo es de los tres.

A mi abuela, porque sé lo orgullosa que te debes sentir de mí.

A Jose y a todos mis amigos, por interesarse por el proyecto y darme ánimos desde el primer día.

A Ana, Ramón, Moya, Álvaro, Toñín, Pablo, Piña, Bruno, Ismael, Antonio, Alberto y Dhivesh, por participar en el experimento del test de Turing.

A los miembros del tribunal, por participar de este momento tan importante para mí.

A mi tutor, Raúl Arrabales, por su dedicación al proyecto y por motivarme constantemente para seguir adelante. Me considero afortunado de haber podido poner el punto y final a la carrera trabajando con un profesor como tú.

Gracias a todos

Índice de Contenido

Índice de Tablas

Índice de Figuras

Índice de Requisitos

Índice de Casos de Uso

1. Introducción

¿Pueden pensar las máquinas? Esta pregunta resultó osada cuando Alan M. Turing (matemático, criptógrafo y filósofo inglés de mediados del siglo XX) la planteó por primera vez. Resulta obvio que a simple vista, este don corresponde a los seres humanos, que han desarrollado durante millones de años, las estructuras celulares que conforman sus complejos cerebros. Ni siquiera podemos afirmar que esta propiedad sea compartida por otras especies de seres vivos, ya que en torno a esa idea existen numerosos estudios que a día de hoy no han sido capaces de alcanzar una respuesta rotunda y concluyente al respecto, puesto que aspectos como la conciencia y el pensamiento animal son temas muy controvertidos a día de hoy.

Pero la respuesta a esta pregunta no es tan sencilla; dependerá, como bien dijo Turing, "de qué entendamos por pensar". Según el Cambridge Advanced Learner's Dictionary [*1*], pensar supone la acción de utilizar el cerebro para planear, solventar o decidir algo. Esta definición se quedaría pequeña si no consideramos que "cerebro" es toda estructura natural o artificial, utilizada para la consecución de los fines descritos anteriormente. Si acudimos a la definición de la Real Academia Española de la Lengua [*2*], pensar supone el acto de "examinar con cuidado algo para formar dictamen". Por tanto, queda una puerta abierta a que las máquinas, valiéndose de técnicas de Inteligencia Artificial puedan considerarse elementos racionales, al ser capaces de evaluar un determinado entorno y actuar en consecuencia.

En 1950, Turing lanzó un desafío, el test de Turing. Su intención era la de corroborar la existencia de inteligencia en las máquinas. La prueba estaba estructurada de forma que un jurado situado en una habitación, se encargaba de formular preguntas a dos competidores (una máquina y un ser humano) situados en otras habitaciones y, basándose en sus respuestas, debía decidir cuál era la máquina y cuál el ser humano. De este modo, se consideraría que el test había sido superado si el jurado no fuese capaz de descubrir la identidad real de cada uno de los participantes.

Desde entonces, han surgido numerosas variantes de este test que persiguen un mismo objetivo, pero con aplicación en diversos campos de investigación. Aún hoy, ninguna máquina ha sido capaz de superar el test con éxito.

La Inteligencia Artificial es uno de los campos de la Informática que está experimentando un mayor crecimiento en los últimos años. Aún así, el ser humano lleva conviviendo con algunas técnicas posteriormente vinculadas a la Inteligencia Artificial, desde mucho antes de ser consciente de ello. Papiros egipcios que datan del 3000 a. C. constatan que en los procesos quirúrgicos de la época, se utilizaban sistemas expertos a modo de guías de operación, teniendo en cuenta determinados síntomas que presentase el paciente.

Asimismo, también se observa que en la actualidad otros campos profesionales como la literatura y el teatro, se han visto inspirados por la

Inteligencia Artificial a la hora de generar obras de culto. Este es el caso de la obra de teatro de Karel Capek de 1921 titulada "Rossum's Universal Robots", con la que se acuñó por primera vez el término "Robot"; Mary Shelley, con su conocido relato "Frankenstein", en el que un científico es capaz de originar vida inteligente mediante un experimento macabro, Isaac Asimov y sus obras literarias relacionadas con el mundo de la robótica, etc.

A pesar de ello, apenas se ha comenzado a explorar (de manera real, tangible) toda la utilidad que las técnicas pertenecientes a esta ciencia son capaces de brindar en entornos tan dispares entre sí, como la Medicina, la Economía, la Robótica, la industria del Software, etc.

Especialmente interesante resulta este último campo ya que supone una herramienta tremendamente multidisciplinar, integrable con infinidad de ámbitos profesionales, que consolidan sus innovaciones tecnológicas gracias a él. Existen un gran número de sectores dependientes de la industria del Software y, en muchas ocasiones, dichos campos se encuentran relacionados entre sí, con lo que se alcanzan entornos colaborativos que favorecen su desarrollo.

Los Videojuegos

La industria del entretenimiento digital es un sector que mueve millones de euros al año en todo el mundo. Datos recientes de la Asociación del Software de Entretenimiento de Estados Unidos [*3*], demuestran que las ventas de videojuegos en 2008 en ese país aumentaron un 22.9%, cuadriplicando los datos de 1996.

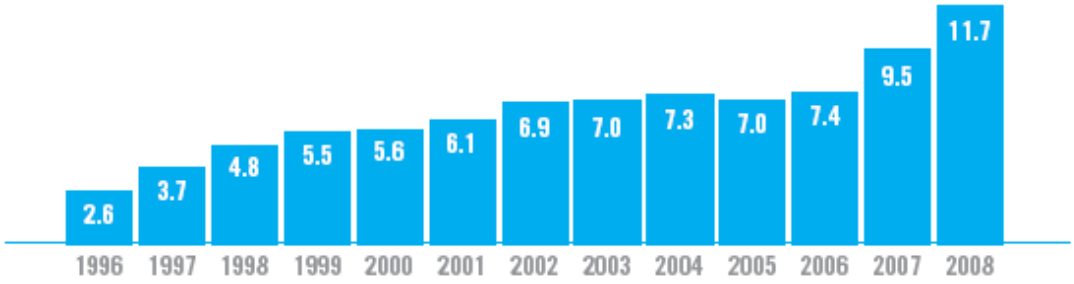

Figura 1 Volumen de ventas de videojuegos en Estados Unidos

En la figura superior se puede observar la trayectoria ascendente que año tras año experimentan las ventas de videojuegos en Estados Unidos.

Los videojuegos, son una de las ramas que aportan una mayor diversidad al sector ya que, más allá de las distintas plataformas existentes (ordenadores y multitud de consolas fabricadas entre otros, por gigantes del sector como Microsoft [*4*], Sony [*5*] o Nintendo [*6*]), también proporcionan un abanico de juegos capaz de satisfacer a cualquier tipo de público. Algunos de estos juegos se pueden encuadrar en las siguientes categorías:

- Aventura. En este género, el usuario (ver *Glosario*) se mete en la piel de un personaje que debe avanzar a través de una o varias tramas que conforman la historia del juego.

- Deportes. Abarcan una gran cantidad de deportes (reales y ficticios) en los que el usuario compite por ser el mejor en diversas competiciones.

- Educativos. Los que tienen como objetivo prioritario potenciar algún aspecto pedagógico. Suelen ir orientados al público más joven. Este tipo es los géneros que utilizan la Inteligencia Artificial en menor medida.

- Estrategia. Son aquellos en los que el usuario se encarga de gestionar un gran número de recursos de forma simultánea para la consecución de los objetivos. La acción de todos los usuarios puede realizarse en tiempo real (todos al mismo tiempo) o por turnos (cada usuario tiene un marco de tiempo para realizar una acción mientras los demás esperan su turno). Suelen ser juegos de gestión (control de parques, hospitales, etc) o bélicos.

- *Shooter* (ver *Glosario*). Juegos de carácter bélico en los que el objetivo del usuario es abrirse camino a través del juego disparando a todo jugador (ver *Glosario*) que se ponga a tiro.

- Lucha. Género en el que los combates cuerpo a cuerpo cobran especial protagonismo.

- Juegos de Mesa. Esta categoría está compuesta por todos aquellos videojuegos encargados de emular los tradicionalmente conocidos como "juegos de mesa", juegos físicos que cuentan con un tablero y fichas (ajedrez, mahjong, etc).

- Party. Este tipo de juegos se centra en el entretenimiento en grupo. Normalmente se utiliza un modo de juego por turnos, en el que gana el usuario que mayor puntuación consiga al cabo de un número determinado de etapas.

- Plataformas. Es una de las categorías más antiguas. El usuario encarna a un jugador que avanza por un mundo en el que debe saltar, correr y alcanzar determinados elementos repartidos por el entorno si quiere alcanzar su objetivo. Los hay tanto bidimensionales como tridimensionales, pero los juegos de culto de este género son del tipo 2D (ver *Glosario*).

- Rol. Probablemente, el género más polémico de todos, dado el nivel de implicación que el usuario puede llegar a alcanzar. En ellos, el usuario representa un papel y debe interactuar con el entorno y el resto de jugadores del videojuego para poder mejorar sus características.

- Simulación. Este tipo de juegos se encargan de reproducir una escena cotidiana (cuidar una mascota, conducir un determinado tipo de vehículo, dirigir la vida de un personaje, etc.).

De entre todos estos géneros, este estudio se centrará en los de tipo *Shooter*, puesto que proporciona un entorno muy acotado, en el que se pueden reproducir diversas técnicas de Inteligencia Artificial con relativa facilidad. La forma de aplicar el test de Turing en un videojuego, será desarrollando y posteriormente evaluando los agentes autónomos (ver *Glosario*), también conocidos como AI Bots o personajes sintéticos, utilizados durante la ejecución. Dentro del juego estos bots desempeñan el papel de jugador no controlado por ningún usuario humano. Es decir, su comportamiento es controlado de forma automática en el juego mediante un programa de Inteligencia Artificial, con el fin de interactuar de algún modo con el jugador/es manejado/s por el usuario. Del mismo modo, se estudiarán las posibles repercusiones a nivel tecnológico que posee el desarrollo de este tipo de agentes.

Objetivos del Estudio

El objetivo principal de este proyecto, tal y como reza su título, es llevar a cabo un estudio de la capacidad cognitiva de los agentes autónomos (bots) utilizados en videojuegos y, más concretamente, en los de tipo *Shooter*. Para ello, se analizarán diversas alternativas de desarrollo que se verán en detalle durante posteriores secciones de este documento.

Asumiendo que, por muy inteligente que resulte el bot a analizar, no superará con éxito el test de Turing, será necesario contar con un mecanismo capaz de valorar el nivel de cognición alcanzado, independientemente del resultado del test anterior. Por ello, se hará uso de la herramienta *ConsScale* [*7*], cuya labor es la de calcular cuán "inteligente" es el bot basándose en una serie de hitos cognitivos evaluados durante su ejecución. Esta herramienta será analizada en detalle durante el apartado de *Estado del Arte*.

Para alcanzar el objetivo fijado, será necesario plantear una serie de hitos a cumplir:

- Configurar un entorno de experimentación. Dicho entorno servirá para llevar a cabo un conjunto de pruebas que nos permitan comprobar la capacidad cognitiva de las soluciones desarrolladas a lo largo de este proyecto,

- Implementación de soluciones. Dichas soluciones serán evaluadas en el entorno de experimentación previamente configurado. En el proceso de desarrollo de estos bots se explorarán diversas técnicas de IA que pueden ser empleadas, tales como el algoritmo Q Learning de aprendizaje por refuerzo, mecanismos de sistemas expertos o estructuras basadas en autómatas finitos.

- Obtener conclusiones sobre la adaptación del test de Turing en plataformas de videojuegos, así como las posibles mejoras que se pueden adoptar de cara a mejorar los resultados obtenidos en dicha prueba.

Estructura del Documento

A lo largo de este documento, se pueden encontrar las siguientes secciones:

- Gestión y Organización del Proyecto. En este apartado se tratarán aspectos relativos a la metodología de trabajo utilizada durante la realización del proyecto.

- Estado del arte. En esta sección se revisarán las últimas innovaciones surgidas respecto al desarrollo de comportamientos artificiales en el mundo de los videojuegos de tipo *Shooter*. Gracias a su estudio, se tendrá una idea del nivel actual existente y contaremos con una base firme desde la que comenzar el estudio.

- Análisis y Diseño de la Arquitectura: En este apartado, se realizará un análisis de la arquitectura software a utilizar durante el desarrollo de este proyecto (plataformas con las que se trabaja, aplicaciones, lenguajes, etc). Del mismo modo, se describirá las técnicas utilizadas durante los procesos de implementación y pruebas de los bots generados.

- Trabajo Realizado. Una vez analizado el entorno de trabajo y definidos los objetivos que se pretenden alcanzar, esta sección contendrá toda la información relativa al desarrollo del proyecto, detallando el proceso seguido para la realización de cada uno de los prototipos de bot implementados.

- Diseño de Experimentos y Resultados. En esta sección se incluirán todos los experimentos realizados para comprobar las ventajas e inconvenientes que presentan cada uno de los prototipos desarrollados durante la fase de desarrollo. Para ello, se formularán hipótesis, seguidas de un conjunto de pruebas y un análisis de resultados que se encargarán de confirmar o rechazar las conjeturas realizadas inicialmente para cada una de las pruebas.

- Conclusiones. En este apartado, se reflexionará acerca de la consecución (o no) de los objetivos globales inicialmente planteados para este proyecto. Además, se estudiarán la repercusión del desarrollo de bots en campos no relacionados con los videojuegos, como la robótica.

- Líneas Futuras. Finalmente, se citarán diversas líneas de investigación que en un futuro y, teniendo como base el proyecto actual, puedan resultar interesantes de cara a ampliar el estudio realizado en el mismo.

- Bibliografía. Relación de todas las fuentes de información consultadas para la realización del proyecto.

- Referencias. Fuentes bibliográficas a las que se ha hecho referencia en el texto.
- Glosario. Listado de términos técnicos utilizados a lo largo de este documento y su correspondiente significado.
- Anexos. Sección que contendrá información adicional del proyecto, referenciada desde apartados anteriores, como porciones de código, capturas de pantalla, etc.

2. Gestión y Organización del Proyecto

A lo largo de esta sección, se tratarán diversos aspectos relacionados con la estructuración del proyecto. En concreto, se hablará de las metodologías y modelos de desarrollo empleados de cara a obtener un resultado final que siga estándares reconocidos y que provean a este proyecto de la máxima calidad en todos los aspectos.

En concreto, para la realización del proyecto se ha seguido la filosofía del estándar PSS-05-0 [*8*] de la Agencia Espacial Europea (ESA) [*9*], pero aplicada a proyectos pequeños. Según este estándar, un proyecto se considera "pequeño" cuando se dan uno o más de los siguientes requisitos:

- Las labores de desarrollo requieren un esfuerzo máximo de dos años hombre.
- El equipo humano involucrado en el proyecto no supera las cinco personas.
- La cantidad de líneas de código implementadas es inferior a 10000 (excluyendo los comentarios).

Es preciso clarificar que en ningún caso se pretende aplicar este estándar de manera estricta dentro del proyecto puesto que, en muchas ocasiones, existen documentos o parte de los mismos que no son aplicables dentro de este marco de trabajo. Por tanto, se busca seguir la misma filosofía pero de una manera ciertamente más flexible, que nos permita amoldar esta metodología de trabajo a nuestras necesidades. A continuación, se detallan los aspectos a aplicar del PSS-05-0 de cara a garantizar una estandarización de este proyecto:

Ciclo de Vida del Software

Para la realización de este proyecto, se llevará a cabo la aplicación de un modelo de ciclo de vida de software que contribuya a la estandarización y calidad de todo el proceso. Se han analizado diversos modelos con el fin de utilizar el más adecuado en nuestro caso. A continuación se detallan las características de cada uno de los modelos analizados y los motivos de haberla aceptado o descartado:

- Modelo en cascada. Como se puede observar en la *Figura 2*, este modelo aboga por una extracción de requisitos del sistema, seguida de un análisis de requisitos software. Posteriormente se realiza un diseño en dos fases (preliminar y detallado), una etapa de codificación y pruebas y, finalmente, una fase de explotación y mantenimiento. Dicho esquema presenta ciertos inconvenientes, ya que se puede invertir una gran cantidad de tiempo en finalizar todo el proceso y además, no se ven resultados en el código hasta la etapa final, lo cual aporta un grado de incertidumbre innecesario al proyecto. Por este motivo, se tomó la decisión de no utilizar este modelo para nuestro estudio.

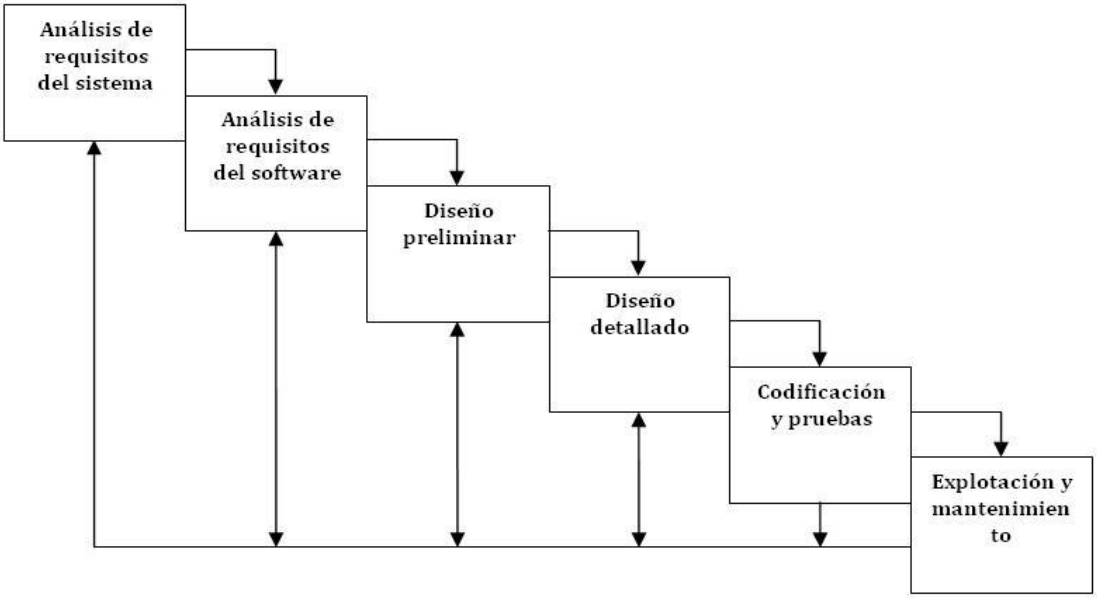

Figura 2 Ciclo de vida en cascada

- Modelo incremental. Este tipo de ciclo de vida se basa en la idea de aportar porciones parciales de código, llamadas incrementos (ver *Figura 3*). Normalmente, cada incremento se fundamenta en la anterior entrega. Se utiliza especialmente cuando el plazo de entrega es inamovible en el tiempo. En nuestro caso, la aplicación de este modelo presentaría una gran ventaja, puesto que se generan porciones de código con mayor antelación que en el caso anterior. Sin embargo, existe un problema derivado de este modelo: los errores en los requisitos se detectan demasiado tarde y además, este modelo no permite regresar a las fases iniciales del proyecto en las que dichas restricciones son definidas. Por tanto, esta alternativa es descartada por su rigidez ante la posibilidad de rehacer determinadas etapas.

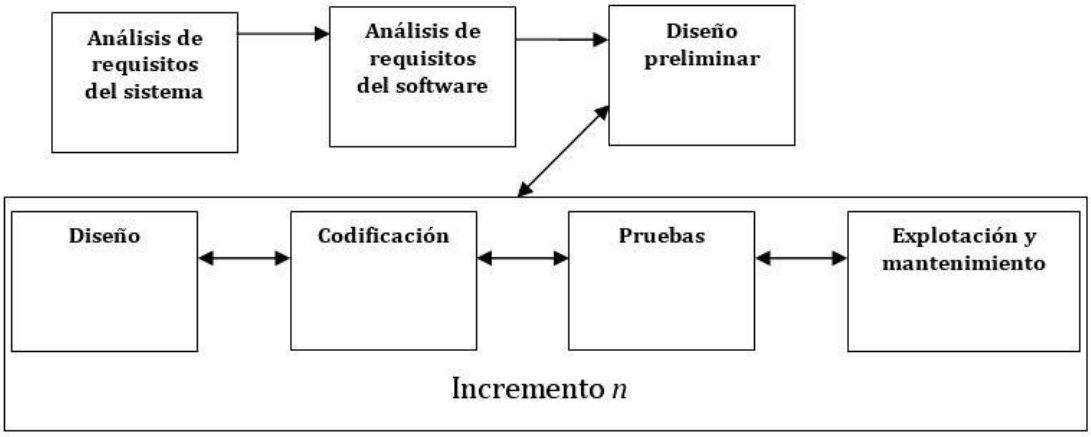

Figura 3 Ciclo de vida incremental

- Modelo en espiral. Como puede observarse en la *Figura 4*, este modelo consiste en realizar una secuencia repetitiva de procesos de forma que, en cada una de las iteraciones se lleve a cabo una planificación, en la cual se recopila información relativa a los requisitos del proyecto; un estudio de riesgos correspondientes a cada una de las distintas

alternativas que cumplen con los requisitos planteados; una fase de codificación, que dará como resultado un prototipo evaluable; y una etapa final de evaluación en la que se comprobará el comportamiento del prototipo y se valorará la conveniencia de realizar otra iteración. Este último modelo es el más versátil de cuantos se han analizado y es el único que incluye labores de análisis de riesgos de forma explícita. Además, permite retroceder a fases previas del proyecto y favorece la eliminación de errores y alternativas poco atractivas al comienzo de cada iteración. Por todo ello, este modelo resulta una opción aconsejable.

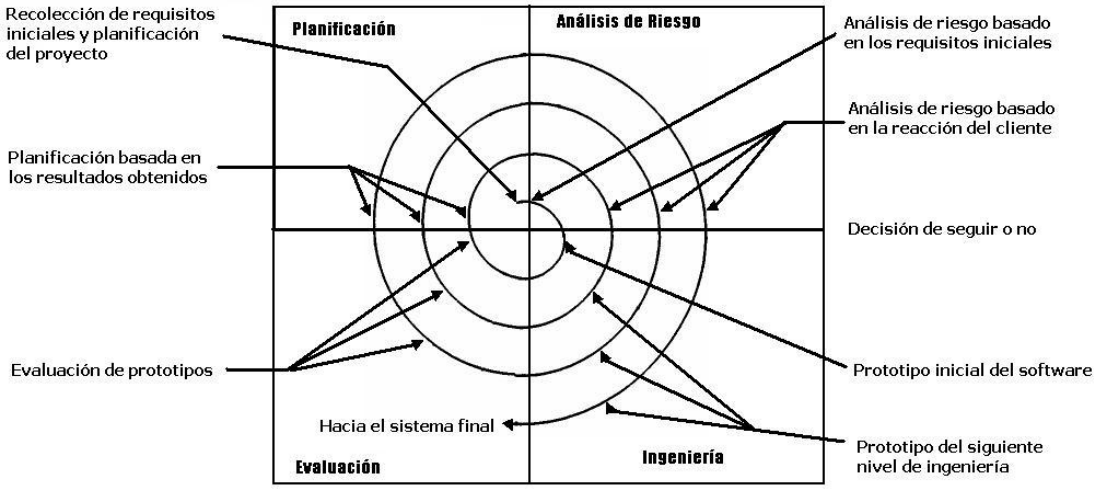

Figura 4 Ciclo de vida en espiral

Finalmente y tras evaluar todas las ventajas e inconvenientes de cada uno de los modelos de ciclo de vida de software propuestos, se optará por incorporar dentro de este estudio la filosofía de trabajo del modelo en espiral ya que gracias al uso de prototipos y a las facilidades que ofrece a la hora de modificar diversos aspectos desde el comienzo de cada iteración, resulta una alternativa apropiada de cara a generar varias versiones sucesivas del comportamiento de un agente autónomo, así como para desarrollar diversas alternativas de un comportamiento base. De este modo, cada una de las versiones estables que se implementen, será considerada un prototipo y, la conveniencia de realizar nuevas iteraciones vendrá impuesta por la eficiencia demostrada por el bot en los resultados de la etapa de evaluación.

Requisitos de Usuario y del Software

Para poder llevar a cabo la aplicación del ciclo de vida en espiral, será necesario abarcar todas y cada una de sus fases. Por ello y, en consonancia con la documentación existente en el estándar PSS-05-0 de la ESA, será preciso aportar una relación de requisitos de usuario que ayuden a clarificar las especificaciones iniciales del proyecto.

Del mismo modo, será necesario aportar información adicional, de carácter técnico, acerca de los requisitos que deben ser satisfechos de cara al desarrollo de este proyecto. De esta manera se hará una división de los mismos

en las diversas categorías a las que hacen referencia. Dichas categorías serán, en el caso de los requisitos de usuario:

- Capacidad. Recoge aquellos requisitos que representan la esencia del sistema. Es decir, qué cosas se deben hacer.
- Restricción. Engloba restricciones inherentes al sistema, que pueden tener que ver con aspectos relativos a la comunicación, adaptabilidad, portabilidad y estabilidad del mismo.

En cuanto a los requisitos del software, se encuadran en numerosas categorías, de entre las que destacan:

- Funcionales. Recoge aquellos requisitos que tienen que ver con el funcionamiento genérico del sistema, pero no con los detalles más técnicos del mismo.
- De rendimiento. Aspectos relativos a los mecanismos utilizados por el sistema para llevar a cabo su actividad de manera adecuada.
- De interfaz. Información relativa a las herramientas provistas para que el usuario interactúe con el sistema.
- De recursos. Requisitos de hardware impuestos por el sistema a desarrollar.
- De portabilidad. Aspectos relativos a la posibilidad de migrar el sistema entre diversas arquitecturas software con el mínimo esfuerzo.
- De calidad. Relativa a los procedimientos a realizar para garantizar la calidad del producto final.
- De estabilidad. Hace referencia a los mecanismos implementados para evitar problemas inesperados durante la ejecución del sistema.

Existen muchas otras categorías que no son especificadas en esta sección, al no ser aplicables en este proyecto.

Cabe destacar que todos los requisitos del documento deben ser verificables; es decir, debe existir un mecanismo para confirmar que realmente se están cumpliendo. Por ello, cada uno de los requisitos de usuario debe estar sustentado al menos, por un requisito del software que defina de forma detallada, la manera de llevarlo a cabo. La generación de los requisitos de software completa la etapa de planificación del ciclo de vida en espiral. En cuanto a la trazabilidad del producto final, se debe proporcionar información suficiente relativa a todos los pasos dados durante el desarrollo del sistema. Por tanto, será necesario conservar todas las versiones realizadas de las distintas soluciones finales implementadas.

La labor de obtener requisitos será abordada extensamente en la sección de *Trabajo Realizado*.

Diseño Arquitectónico y Detallado

Una vez se ha realizado la recogida de requisitos, es el momento de aportar alternativas que los satisfagan. Para ello, se propondrán diversas soluciones, especificando el diseño arquitectónico de cada una de ellas, así como aspectos relativos a sus correspondientes diseños detallados, identificando los componentes principales del sistema, así como el flujo de información existente entre ellos.

En el caso del diseño arquitectónico, se debe aportar la siguiente información sobre el comportamiento del bot (sistema):

- Datos de entrada al sistema.
- Datos intermedios del sistema.
- Datos de salida del sistema.
- Relación entre los principales componentes.

El diseño detallado se encargará de ahondar en todos los aspectos relativos al diseño arquitectónico que precisen de un grado mayor de explicación. Es el caso de la estructura de componentes, la cual debe ser desglosada a más bajo nivel.

Al igual que ocurría con los requisitos de usuario, los de software deben ser verificables. Para garantizar este aspecto en este punto, será preciso comprobar que todos y cada uno de los requisitos de software son satisfechos por la funcionalidad de, al menos, un componente de la arquitectura diseñada.

Finalmente y como paso previo a la implementación de un prototipo, será necesario optar por una de las alternativas propuestas durante esta fase (en caso de haber más de una), sopesando las características de cada una de ellas. Tras esta etapa, se da por finalizada la fase de análisis de riesgos del ciclo de vida en espiral, habiendo aplicado los documentos relativos al diseño arquitectónico y diseño detallado correspondientes al estándar PSS-05-0 de la ESA.

Los casos particulares correspondientes a este apartado, serán analizados en la sección de este documento llamada *Trabajo Realizado*.

Implementación y Transferencia del Software

A lo largo de esta etapa, se llevará a cabo la labor de codificación del comportamiento del bot, que será utilizado a modo de prototipo durante la fase de desarrollo y pruebas del modelo en espiral de este proyecto.

Además del desarrollo del código del agente y antes de poder pasar a la fase de evaluación de los resultados del mismo, será preciso llevar a cabo un proceso de transferencia del software mediante el cual, se aplica una batería de pruebas de aceptación al código con el fin de comprobar si éste cumple con todos los requisitos previamente especificados y asegura que el prototipo cumple con los mínimos de calidad impuestos.

Mediante esta tarea, se abarca la tercera fase del ciclo de vida en espiral, la evaluación. Asimismo, se consolida el uso del estándar PSS-05-0 de la ESA, mediante la aplicación conceptos procedentes de los documentos relativos a planes de verificación y validación, y a los que hacen referencia a la garantía de calidad del software.

Toda la información relativa a la aplicación práctica de estos conceptos, está reflejada en el apartado *Trabajo Realizado* de este documento.

Evaluación de Resultados

Este apartado hace referencia a la última fase del modelo en espiral del ciclo de vida del software. En ella, se aplicará el prototipo previamente generado en un entorno de juego de tipo *Shooter* con el fin de obtener unos resultados que nos permitan valorar, de forma cuantitativa, el rendimiento alcanzado por el bot. De los resultados obtenidos en esta fase, dependerá la posibilidad de realizar más iteraciones a través del modelo en espiral.

Entre otras pruebas, se llevará a cabo una aplicación del test de Turing en un entorno de videojuegos, basada en la adaptación de dicha prueba, realizada en la competición BotPrize [*10*], en la cual un jurado se encarga de evaluar si el comportamiento de los participantes en una partida de tipo *Shooter* es propio de un humano o no.

Toda la información relativa a la evaluación de prototipos se puede encontrar en la sección llamada *Diseño de Experimentos y Resultados* de este documento.

3. Estado del Arte

A lo largo de esta sección, se realizará un análisis de los avances y últimas novedades llevados a cabo en cada uno de los aspectos vinculados con este estudio. Dichos aspectos son los siguientes:

- Variantes de juego de tipo *Shooter* más importantes. Se explicarán sus principales características, así como la conveniencia (o no) de usar cada una de ellas en este estudio.

- Evaluación de la capacidad cognitiva de un bot. Se llevará a cabo el análisis de las herramientas disponibles de cara a sopesar la capacidad cognitiva de los agentes autónomos que se desarrollen durante las distintas iteraciones del modelo en espiral.

- La Inteligencia Artificial aplicada en el mundo de los videojuegos. En este apartado se analizará la repercusión que han tenido las técnicas de Inteligencia Artificial aplicadas a la industria del videojuego durante los últimos años y, en concreto, en los juegos de tipo *Shooter*.

- Desarrollo de Bots en juegos FPS. En este apartado se valorarán las alternativas existentes a la hora de desarrollar un agente inteligente que pueda ser posteriormente aplicado a un juego de tipo *Shooter*.

A medida que vaya avanzando el estudio sobre el estado del arte de la materia, se irán tomando decisiones que permitan determinar todos los detalles acerca de la plataforma de trabajo que se utilizará durante el resto del proyecto. Es decir, se tomarán decisiones acerca del tipo de categoría dentro de los juegos *Shooter* con el que se trabajará, el juego a utilizar, la técnica de Inteligencia Artificial a aplicar y las herramientas de desarrollo de bots a emplear.

Juegos de Tipo Shooter

Dada su eminente orientación a realizar partidas en red entre varios usuarios, los juegos de tipo *Shooter* comparten una estructura muy característica. Utilizan una arquitectura cliente-servidor para llevar a cabo la comunicación y sincronización de los distintos terminales conectados a la partida. De este modo, el servidor de una partida será el encargado de controlar aspectos genéricos del juego, como la posición de cada jugador o de elementos específicos del escenario, el envío de mensajes entre los distintos usuarios, etc. Cada cliente representará el terminal de un usuario específico que participe en la partida, y que solicite información sobre el estado de la misma al servidor de forma reiterada.

Como se explicó previamente en la sección introductoria [*1*] de este documento, los *Shooter* son juegos de carácter bélico en los que el objetivo del usuario es utilizar la violencia para la consecución de sus fines. Dentro de esta categoría, hay un sinfín de clasificaciones posibles. Si se realiza una clasificación

por el tipo de vista que tiene el usuario de la acción del jugador, existen varias subclases:

- First Person *Shooter* (FPS). Juegos en los que el usuario maneja a un jugador en primera persona (cámara subjetiva). Como se puede observar en la *Figura 5*, este tipo de cámara involucra mucho más al usuario dentro de la acción realizada por el jugador al que controla.

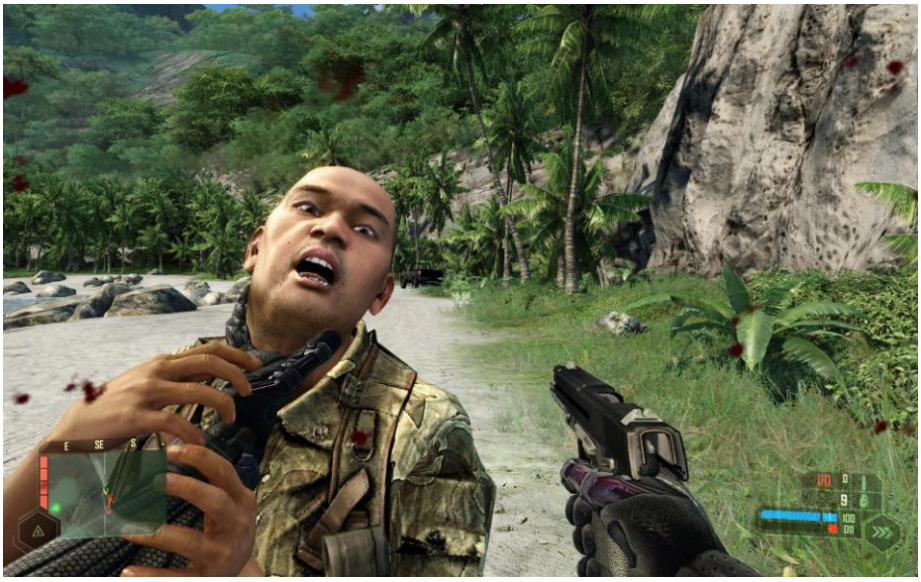

Figura 5 Perspectiva de juego FPS

- Third Person *Shooter* (TPS). Juegos en los que el usuario maneja a un jugador desde un punto de vista externo al mismo (tercera persona). Esta cámara suele permitir personalizar diversos parámetros relativos a la altura y distancia de la cámara respecto al jugador (ver *Figura 6*).

Figura 6 Perspectiva de juego TPS

- Modo híbrido. Permite conmutar entre los dos modos anteriores, abarcando así las preferencias de todos los usuarios.

Figura 7 Distintas perspectivas de un modelo híbrido

El tipo de vista no es un factor determinante a la hora de llevar a cabo un proyecto centrado en aplicar técnicas de Inteligencia Artificial, por lo que el tipo de cámara a utilizar en nuestro estudio es irrelevante y dependerá del juego concreto que se elija.

Además de ésta, también se puede llevar a cabo una clasificación por el tipo de enfoque del juego. Según este criterio, algunos de los principales tipos de juegos *Shooter* son:

- Estrategia militar. Este tipo de juegos, habitualmente basados en hechos reales o grupo militares de reconocido prestigio a nivel mundial, reproducen misiones en las que se encarna a un jugador que, siguiendo tácticas militares debe alcanzar unos objetivos. Ejemplos claros de este tipo de juegos son los pertenecientes a la saga Call of Duty [*11*] o Ghost Recon [*12*]. Tienen una alta carga argumental dado el realismo del que se pretende dotar al juego. Sin embargo, dicho hilo argumental puede reducirse al mero hecho de desarrollar combates entre equipos enemigos en un entorno controlado. Este es el caso del conocido mod (ver *Glosario*) de la saga Half Life [*13*], Counter Strike [*14*], en el cual equipos de terroristas se enfrentan a las fuerzas especiales de la policía sin otra finalidad que derrotar al adversario.

- Juegos online multijugador masivos (MMOG). Pese a que suelen incorporar modalidades de juego offline (ver *Glosario*), este tipo de juegos se especializan en realizar combates a través de una red de usuarios (LAN, Internet, etc). Se realizan batallas sobre escenarios controlados y, adicionalmente, pueden contar con unas normas y tiempo máximo de juego estipulados con antelación. El hilo argumental suele ser tan limitado que en alguno de estos juegos, todo jugador que muere, resucita automáticamente en otro punto del escenario para evitar que el usuario que lo controla quede esperando para volver a jugar hasta que finalice la partida actual. Los representantes más importantes de este tipo de juegos son la saga de videojuegos Unreal Tournament [*15*] y su competidora Quake [*16*].

Para determinar cuál de los dos tipos anteriores de juego es más conveniente de cara a realizar nuestro estudio, es importante tener en cuenta el grado de reutilización que permiten cada una de las alternativas. En el caso de los juegos de estrategia militar, la mayoría de ellos son productos comerciales cuyo código no es liberado y para los cuales no existen mecanismos que permitan agregar agentes autónomos diseñados por el usuario.

En cuanto a los juegos de tipo MMOG, son juegos que persiguen principalmente su uso a través de la red y que, en algunos casos, facilitan las herramientas necesarias para modificar el código fuente o añadir nuevos módulos que aporten funcionalidad adicional sin hacer transparente el código a los usuarios. Por tanto, esta familia de juegos será la elegida para este proyecto al aportar las estructuras necesarias para poder adaptar el entorno de juego a nuestras necesidades en cada momento.

Este tipo de juego restringe casi por completo el modo de vista al de FPS (vista subjetiva o en primera persona), puesto que al tener un menor ángulo de visión, mejora la velocidad de ejecución del juego y es el preferido para los juegos de tipo *Shooter* online (ver *Glosario*), en los que el flujo de datos que deben intercambiar el servidor y los clientes del juego, supone una restricción en términos de rendimiento.

Evaluación de la Capacidad Cognitiva

En el campo de la evaluación cognitiva de agentes inteligentes existe una gran carencia de herramientas ya que, a día de hoy, no existe una escala ampliamente aceptada y consensuada para llevar a cabo labores de cuantificación de dicha cualidad.

La herramienta ConsScale, aunque enfocada en capacidades cognitivas asociadas con la conciencia, se puede utilizar, no sólo para evaluar, sino también para comparar la capacidad cognitiva de distintos agentes entre sí. Posee 13 niveles de cognición, los cuales incluyen siempre a los de nivel inferior, de forma que un agente no pueda pertenecer a un nivel *n* si no forma parte de los *n-1* niveles previos. Dichos niveles son (en sentido ascendente de complejidad):

1. Etéreo. El agente sólo cuenta con su entorno como elemento fundamental.

2. Aislado. El agente posee una estructura (real o simulada). Opcionalmente también puede contar con un conjunto de sensores (propioceptivos y/o exteroceptivos).

3. Descontrolado. El agente cuenta con un conjunto de actuadores para interactuar con entorno.

4. Reactivo. El agente posee mecanismos necesarios para realizar labores coordinadas entre sensores y actuadores.

5. Adaptativo. El agente posee memoria por lo que, combinada con el comportamiento reactivo puede adaptar su comportamiento de forma dinámica a una situación concreta.

6. De atención. El mecanismo de atención permite que se pueda centrar la atención en aspectos concretos de la información percibida o memorizada.

7. Ejecutivo. Se pueden realizar múltiples tareas de manera intercalada, aportando mayor tiempo y esfuerzo a las que tengan mayor prioridad.

8. Emocional. Este nivel representa la capacidad de desarrollar un mecanismo de aprendizaje emocional complejo. Es decir, consiste en ser capaz de conocer el estado emocional del propio agente y del resto de agente con los que interactúe.

9. Auto-Cognitivo. Consiste en que el agente sea capaz de reconocerse a sí mismo.

10. Empático. Implica la capacidad de reaccionar ante situaciones vividas por otros agentes, como si fuesen sufridas por él mismo.

11. Social. En este nivel, el agente debe ser capaz de aplicar comportamientos sociales. Especialmente interesante resulta la posibilidad de incorporar comportamientos como el liderazgo en estudios futuros sobre el uso de los agentes inteligentes en videojuegos.

12. Humanoide. Este nivel implica el desarrollo de comportamientos sociales complejos y el uso de la comunicación verbal.

13. Súper-Cognitivo. Este último nivel aporta la posibilidad de sincronizar y coordinar diversos niveles de conciencia de forma simultánea dentro de un mismo agente.

Para el cálculo de estos niveles, la aplicación proporciona dos secciones en las que el usuario debe seleccionar las características que cumple el agente artificial, tanto a nivel arquitectónico (componentes sensitivos y motores y otros mecanismos que presente el agente) como cognitivo (comportamiento que presenta en determinadas situaciones). Podría considerarse que la evaluación de la arquitectura del agente establece una escala de medición genérica, mientras que la escala de las habilidades cognitivas es más específica al determinar qué características cumple el agente dentro de un determinado nivel arquitectónico. Algunos de los componentes pertenecientes a la evaluación arquitectónica del bot son:

- Entorno. Por defecto, se asume que hasta la arquitectura más elemental posee un entorno con el que interactuar. Si sólo se cuenta con esta característica, la arquitectura del bot se cataloga como “etérea”.

- Cuerpo físico o simulado mediante software. Esta característica se cumple en la mayoría de casos y su consecución implica la calificación arquitectónica del agente como “aislado”, dado que pese a tener un cuerpo (real o simulado), no cuenta con mecanismos que le hagan interactuar con el medio.

- Sensores propioceptivos. Son aquellos que se encargan de medir valores propios del sistema agente.

- Sensores exteroceptivos. Sensores que se encargan de medir diversos aspectos relativos al entorno en que se encuentra el agente.

- Actuadores. Permiten la interacción del agente con su entorno. El uso de este tipo de mecanismos aporta al robot el nivel arquitectónico catalogado como "descontrolado", puesto que aún no se han establecido mecanismos para que el agente haga un uso racional de sus herramientas.

- Coordinación entre sensores y actuadores. Dicha relación entre ambos componentes dota al agente de un mayor grado de precisión al realizar determinadas acciones, en las que la acción a realizar dependa en gran medida de los estímulos percibidos del medio. Alcanzar este aspecto de la arquitectura implica que el agente sea catalogado como "reactivo".

- Memoria. La combinación del comportamiento reactivo y el uso de memoria por parte del agente, favorecen que éste sea capaz de recordar cómo se desencadenaron los hechos en el pasado y pueda evaluar si es conveniente o no incurrir en las mismas situaciones nuevamente. Por tanto, este nivel implica que el agente sea "adaptativo".

La consecución de estos primeros niveles, favorece alcanzar una arquitectura más compleja, en la que se puedan llevar a cabo labores de representación múltiple de memoria, mecanismos de expresión oral, etc.; cada una de las cuales implicaría una grado mayor de complejidad arquitectónica.

Asimismo, existe un coeficiente que permite cuantificar numéricamente los resultados de la evaluación. Este índice es el Resultado Cuantitativo de ConsScale (CQS). Su uso facilita la realización de comparativas, así como la posibilidad de usar dicho valor como función de fitness ante el hipotético uso de técnicas de computación evolutiva sobre el comportamiento del agente.

Existen método similares al utilizado por ConsScale de cara a realizar funciones de evaluación cognitiva, pero están diseñados para ser aplicados en humanos, como por ejemplo el test Minimental [*17*], usado para valorar el deterioro cognitivo en personas mayores. Consiste en una serie de preguntas muy específicas que se realizan al paciente y que son puntuadas dependiendo de si éste las contesta correctamente o no. Se evalúan aspectos tales como la orientación temporal y espacial, memoria inmediata y a corto plazo, cálculo, atención y lenguaje. La puntuación obtenida determina si la capacidad cognitiva es normal, existe una sospecha patológica, un deterioro claro, o incluso un proceso relacionado con la demencia.

Obviamente, el único de los métodos que se puede utilizar de referencia de cara a este estudio, es el de ConsScale, puesto que sólo éste permite evaluar la

capacidad cognitiva de agentes artificiales. En cuanto al nivel que se pretende conseguir con el desarrollo de bots a lo largo de este proyecto, no existe un límite superior fijado de antemano, dado que será mucho mejor cuanto mayor sea el nivel que se consiga alcanzar dentro de esta escala. Sin embargo, será preciso fijar un nivel mínimo a superar de cara a considerar que un prototipo es válido para este estudio. En este caso, se considera que, alcanzar al menos el nivel reactivo es una opción interesante, dado que el agente debe ser capaz, al menos, de reaccionar ante estímulos de su entorno, como disparos enemigos; o de su propio sistema, como un nivel bajo de salud, munición, etc.

De manera adicional, se perseguirá alcanzar el siguiente nivel de complejidad (nivel adaptativo), que garantizará el poder llevar a cabo comportamientos más complejos e inteligentes, haciendo uso de un sistema de memoria.

Finalmente, respecto a los antecedentes en el ámbito de la adaptación del test de Turing al mundo de los videojuegos, existe una competición anual llamada BotPrize, que aplica dicha prueba en un entorno multijugador del videojuego Unreal Tournament 2004 (UT04, en adelante). Dicha adaptación implica que un jurado examine una partida multijugador en la que participan diversos usuarios humanos y agentes sintéticos de manera anónima y evalúen cuán humanos resultan sus comportamientos. Si el 80% del tribunal considera que un bot es controlado en tiempo real por un humano, dicho bot será el ganador del certamen. No obstante, dado que esta situación no se ha dado nunca, el primer clasificado recibe una recompensa económica por su labor.

La Inteligencia Artificial en Videojuegos Comerciales

Durante la década de los 90, comenzaron a utilizarse, aunque de forma muy rudimentaria, técnicas vinculadas a la Inteligencia Artificial (IA), con el objetivo de desarrollar videojuegos mucho más complejos que los que había hasta entonces. Esta necesidad venía acuciada por los juegos enormemente predecibles que había en aquella época, en los que las acciones realizadas por los jugadores no controlados por el usuario o NPCs (ver *Glosario*), se repetían con cierta asiduidad. Videojuegos como Herzog Zwei [*18*] o Dune II [*19*] comenzaban a aplicar máquinas de estados para poder desarrollar estrategias en tiempo real. Del mismo modo, en 1996 el videojuego Battlecruiser 3000AD [*20*] utilizó por primera vez redes de neuronas artificiales.

En la década de los 90, existe un juego que, por la naturaleza de este estudio, merece una mención especial. Este videojuego es Golden Eye [*21*] (1997), basado en las aventuras del personaje cinematográfico James Bond, al convertirse en el primer juego de tipo FPS que aplicaba técnicas de Inteligencia Artificial durante el juego. De este modo, los NPCs reaccionaban convenientemente al movimiento del jugador controlado por el usuario. Aún así, este juego presentaba un gran inconveniente: los enemigos del usuario sabían en todo momento dónde se encontraba éste, incluso si aún no lo habían visto, con lo que era imposible esconderse para desarrollar acciones más reales de comportamiento.

Ya en el siglo XXI, los videojuegos aplican técnicas de inteligencia artificial de manera más compleja, con fines tan dispares como detectar elementos que impliquen una amenaza, como en el caso de Halo [*22*] (2001); aplicar estrategias de ataque grupal basadas en el comportamiento del jugador controlado por el usuario, como en FarCry [*23*] (2004) y F.E.A.R. [*24*] (2005) o dotar al juego de un mayor grado de realismo, haciendo que los NPCs vivan sus vidas de manera totalmente independiente, como en The Elder Scrolls IV: Oblivion [*25*] (2006). Todas estas líneas de trabajo persiguen lograr una mayor credibilidad de los roles desempeñados por los distintos agentes autónomos de los videojuegos, aunque esto es un reto aún pendiente de alcanzar.

Los anteriores ejemplos corresponden a juegos con carácter comercial que, en el mejor de los casos, llegan a desvelar en qué aspectos aplican técnicas de Inteligencia Artificial. No obstante no se proporciona información detallada respecto a la técnica en cuestión, por lo que únicamente se pueden emitir conjeturas respecto a cuáles son las que, con mayor probabilidad, se han aplicado en cada caso. Sin embargo, existen juegos que a pesar de haber tenido una gran aceptación popular, mantienen su código abierto a todos los usuarios o, al menos, facilitan herramientas para poder personalizarlos, de manera que sea mucho más sencillo comprender el funcionamiento interno del mismo. Este es el caso de las sagas de juegos Unreal Tournament y Quake descritas anteriormente en el apartado de *Juegos de Tipo Shooter*. Este aspecto nos lleva a centrar más la atención en la utilización de una de estas dos familias de videojuegos en el desarrollo de este proyecto ya que, al menos, facilitan mayor cantidad de información respecto a las técnicas implementadas por el entorno de juego y/o los bots implementados y, por tanto, proporcionan un estado del arte más sólido. En el siguiente apartado de esta sección (ver *Desarrollo de Bots en Juegos FPS*) se analizarán, entre otras, las características de ambas alternativas, se sopesarán las ventajas e inconvenientes de cada una de ellas y se elegirá la que más facilidades ofrezca de cara a un desarrollo de agentes autónomos transparente para el usuario.

Técnicas a emplear en el desarrollo de comportamientos

En esta sección se explicará en detalle el funcionamiento de las diversas técnicas que finalmente serán aplicadas a lo largo del proceso de desarrollo.

Sistemas expertos

Los sistemas expertos son una técnica de IA que suponen un mecanismo de gestión del conocimiento, mediante el cual, toda la experiencia de un conjunto de expertos en la materia a tratar, es procesada y almacenada en un sistema que será utilizado posteriormente para la resolución de problemas.

La definición formal de un sistema experto, implica contar con los siguientes elementos en el sistema:

- Base de conocimiento. Contiene el conocimiento extraído de las conversaciones con los distintos expertos implicados en la realización del sistema.

- Base de hechos. Contiene los hechos relativos a un problema que se han descubierto durante la fase de análisis.

- Motor de inferencia. Modela el proceso de razonamiento humano.

- Módulos de almacenamiento y adquisición de conocimiento. Encargados de aportar nuevos conocimientos al sistema, así como de devolver convenientemente los almacenados con anterioridad.

- Interfaz de usuario. Componente encargada de la interacción hombre-máquina.

Como se verá posteriormente en el apartado de *Ciclo 1: Desarrollo mediante Sistemas Expertos*, el sistema experto a utilizar para el desarrollo del primer prototipo del estudio, será adaptado de forma que se pueda utilizar el conocimiento almacenado, de una forma mucho más dinámica. De este modo, no será necesario definir estrictamente los componentes arriba descritos.

Aprendizaje por refuerzo

El aprendizaje por refuerzo constituye la segunda de las técnicas de IA que se abordarán durante el desarrollo de este proyecto. Los distintos algoritmos de aprendizaje por refuerzo persiguen hallar una política que establezca la conveniencia de realizar una actividad cuando el sistema (en este caso el agente autónomo) se encuentra en una situación concreta. En este estudio, nos centraremos en un algoritmo concreto de esta técnica: *Q Learning*.

El algoritmo de *Q Learning* se encarga de aplicar una función que ayuda a deducir la utilidad de realizar una determinada acción cuando el estado se encuentra en un estado concreto. Su mayor ventaja es la capacidad de concluir la conveniencia (o no) de recalar en una determinada situación, sin necesidad de tener conocimiento previo del entorno de trabajo.

Según reza este algoritmo, se tienen los siguientes elementos:

- Un conjunto de estados *S*. Como se verá en la sección de *Trabajo Realizado*, será importante llevar a cabo una correcta definición y discretización de los estados a utilizar a lo largo del estudio, con el fin de no incurrir en problemas de utilización excesiva de recursos.

- Un conjunto de acciones *A.* En este caso, el conjunto de acciones estará compuesto por cada una de las actividades que realizará el agente autónomo a lo largo del juego.

- Un agente encargado de realizar las acciones y que transitará entre los distintos estados existentes. Dicho agente será representado por el bot a modelar durante los diversos ciclos de la sección de *Trabajo Realizado*.

Cuando el agente realiza una acción del conjunto *A*, es susceptible de pasar de un estado a otro. Cada estado proporciona al agente una recompensa o un castigo dependiendo del desenlace de la acción, al que se llamará “refuerzo”. El objetivo del agente será maximizar el refuerzo correspondiente a cada

relación acción-estado existente en el sistema. A la calidad del refuerzo de una relación acción-estado se le denomina *Q* e inicialmente tendrá un valor nulo (0) para cada una de las relaciones. Cada vez que el agente recibe un refuerzo tras un cambio de estado, los nuevos valores *Q* son recalculados y almacenados en una tabla. Por tanto, la esencia del algoritmo consiste en la actualización de los valores *Q* de la tabla de manera iterativa. La función de cálculo es la siguiente:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t(s_t, a_t)(r_t + \chi \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Donde:

- s es un estado concreto del conjunto *S*.
- a es una acción concreta del conjunto *A*.
- $Q(s_t, a_t)$ representa el valor *Q* correspondiente a una relación estado-acción concreta dada en un instante *t* de tiempo.
- r representa el refuerzo aplicado.
- $\alpha_t(s_t, a_t)$ es la tasa de aprendizaje correspondiente a una relación concreta. Es decir, el índice que determina la importancia del refuerzo a aplicar, de cara al proceso de aprendizaje del agente. De este modo, una tasa de aprendizaje próximo a 0, implica que el agente no aprenda nada, mientras que una tasa de 1, conlleva que el agente sólo tenga en cuenta la información más reciente. El rango de valores válido de alpha va de 0 (no incluido) a 1 (inclusive).
- χ es el factor de descuento aplicado al cálculo o lo que es lo mismo, la tasa que determina la importancia de futuros refuerzos. Un valor próximo a 0, denota que el agente sólo tendrá en cuenta los refuerzos actuales en cada momento, mientras que un descuento próximo a uno implicará que se realicen refuerzos prolongados en el tiempo, con lo que se valorará no sólo la última situación que condujo al agente al estado actual, sino la situación previa a esa, etc. El rango de valores válido de gamma va de 0 (inclusive) a 1 (no incluido).

Los valores a utilizar para las constantes alpha y gamma no siguen un patrón concreto. La idoneidad de dichos valores dependerá del problema a resolver, y será preciso probar varias configuraciones para alcanzar alguna conclusión al respecto. Los valores definitivos deben alcanzar un compromiso entre la capacidad del sistema de explorar nuevas estrategias que reporten mayores bonificaciones y la explotación de las ya exploradas que tengan resultados satisfactorios (aún sin saber si son los óptimos).

Existen otros algoritmos de aprendizaje por refuerzo, como *SARSA* y *Time Difference* que, tras haber sido analizados, finalmente no se han utilizado a lo largo de este estudio.

Autómatas de estados finitos

También conocidos como máquinas de estados finitos o autómatas finitos, estos mecanismos no constituyen una técnica de IA, pero son fácilmente integrables con muchas de ellas. Un autómata finito representa un modelo de comportamiento, compuesto por un conjunto finito de nodos (o estados) unidos entre sí mediante enlaces. Cada nodo de la máquina de estados representa un comportamiento concreto y, para transitar entre dos nodos del autómata, es necesario satisfacer las condiciones prefijadas para el enlace que los comunica.

Como se verá posteriormente en las secciones *Ciclo 4: Desarrollo mediante Sistema Híbrido* y *Ciclo 5: Evolución del Prototipo del Ciclo 4*, cada nodo del autómata aplicado al comportamiento del prototipo en cuestión, implicará la realización de una acción concreta dentro del juego. Del mismo modo, para transitar entre los diversos nodos del autómata, será necesario cumplir con los requisitos relativos al estado del agente, que marquen cada uno de los enlaces.

Desarrollo de Bots en Juegos FPS

A lo largo de este apartado, se llevará a cabo un estudio sobre algunos de los videojuegos de tipo FPS más populares que proporcionan mecanismos para personalizar en mayor o menor medida el comportamiento de los NPCs que intervienen durante las partidas. Los elegidos son los siguientes:

- Counter Strike. Esta familia de videojuegos representan el mod más popular de cuantos han derivado de las distintas entregas del prestigioso videojuego de acción en primera persona Half Life. En esta modificación del código original, el objetivo fundamental es desarrollar combates entre fuerzas terroristas y antiterroristas a través de un entorno controlado y por un tiempo definido con antelación. Esta alternativa, proporciona un kit de desarrollo de bots mediante waypoints (ver *Glosario*). Dichos elementos se encargan de establecer un mapa de puntos dentro del escenario, por los que se define el movimiento del agente. Por tanto, se podría considerar un camino "de migas de pan" que el bot puede seguir en su recorrido constante del escenario.

- Quake. Esta saga de juegos desarrollados bajo licencia GPL [*26*] (ver *Glosario*) ofrece una visión futurista de los juegos de acción en primera persona. Al contar con código abierto, los usuarios pueden aportar mejoras al código. Respecto al desarrollo de bots, esta plataforma ofrece un lenguaje propio llamado QuakeC [*27*], que permite añadir agentes creados por el usuario. No obstante, desde el lanzamiento de Quake 2, para el cual surgieron diversas alternativas de bots inteligentes, ninguna otra versión de esta saga ha aportado herramientas de desarrollo de agentes más allá de aspectos relativos a su comportamiento conversacional y a sus preferencias. Es decir, puede especificarse cuáles son las armas favoritas del agente, y qué comentarios puede hacer en determinados momentos del juego, pero no se pueden aplicar técnicas de Inteligencia Artificial a las acciones que realice durante una partida, puesto que estas son fijas.

- Unreal Tournament. Esta familia de juegos FPS supone la competencia más directa de Quake. No obstante existen diferencias sustanciales que hacen que esta alternativa sea mucho más aconsejable de cara a desarrollar este estudio. La saga Unreal cuenta con una temática idéntica a la de su rival pero, ésta antepone aspectos relativos a la jugabilidad a la calidad gráfica. Sin ir más lejos, las armas de Unreal Tournament ofrecen una doble función (un disparo primario, y otro alternativo, normalmente más letal e impreciso) en todas sus armas, lo que abre un abanico de estrategias de disparo en función de la distancia y nivel de los adversarios. Además, pese a contar con licencias comerciales, las distintas entregas de Unreal Tournament ofrecen plataformas de desarrollo de agentes autónomos. Concretamente en UT04 (la penúltima entrega de la colección) se desarrolló con éxito un proyecto conocido como Pogamut [*28*], que se encarga del desarrollo y depuración de agentes autónomos aplicables sobre uno de los mods del juego, llamado Gamebots [*29*].

Una vez analizadas las tres plataformas de juego propuestas para el desarrollo de este proyecto, será necesario evaluar las ventajas e inconvenientes de cada una de ellas. Se realizará un resumen de las mismas en la tabla siguiente:

	Gratuito	**Código abierto**	**Permite bots**	**Permite AI bots**	**Herramientas para desarrollo de AI bots**
Counter Strike	No	No	Sí	Sí, pero sólo define caminos a recorrer	Waypoints
Quake	Sí	Sí	Sí	No en versiones recientes del juego	---
Unreal Tournament	Sí, pero sólo por terminal	No	Sí	Sí	Pogamut a través de Gamebots

Tabla 1 Comparativa entre plataformas de videojuegos

Como se puede comprobar en la *Tabla 1*, tan sólo dos de las tres alternativas nos permiten desarrollar algún tipo de estrategia en la que intervengan las técnicas de IA para mejorar el comportamiento de los bots. Por tanto, será necesario elegir entre una de las dos plataformas, de cara a poder llevar a cabo el resto del estudio. El uso de bots para el videojuego Counter Strike plantea un inconveniente, ya que permite establecer recorridos por los que se moverá el agente durante una partida real, pero las labores de ataque son automáticas (es decir, el usuario no puede definir de forma explícita qué se debe hacer cuando el agente encuentre a un enemigo, porque su implementación le

obligará siempre a disparar). Por el contrario, el mod Gamebots de Unreal Tournament, establece al comienzo de cada partida, los mecanismos necesarios para que un bot sea capaz de recorrer el escenario (es un concepto similar al de los waypoints, pero en esta ocasión, vienen definidos para cada escenario y no es necesario que el usuario los especifique para cada bot y mapa concretos). Además, su comportamiento puede ser definido como un conjunto de métodos Java, por lo que permite aplicar un gran número de técnicas de IA a su comportamiento.

Por todo ello, se utilizará UT04 y, en concreto, la plataforma de desarrollo de bots Pogamut para implementar las soluciones de este proyecto.

Unreal Tournament 2004

A lo largo de este apartado de describirán todos los aspectos relativos al entorno de juego a utilizar durante el estudio.

Modos de Juego

Una vez decidida la plataforma de juegos a emplear en el estudio, es importante decidir cuál de las distintas modalidades de juego es la que se utilizará. Los formatos existentes son los siguientes:

- *Capture the Flag* (Captura la Bandera). Modo de juego por equipos en el que cada grupo tiene una base y una bandera que defender. El objetivo es robar la bandera del enemigo y llevarla hasta la base de equipo contrario.

- *Deathmatch* (Todos contra todos). Modo de juego individual en el que gana el jugador que más muertes registre en un tiempo determinado o que alcance un nivel predefinido de muertes de jugadores rivales.

- *Team Deathmatch* (Combate por equipos). En este modo, las reglas son las mismas que en la modalidad *Deathmatch*, pero en este caso el marcador será la suma de los resultados individuales de todos los miembros del equipo.

- *Double Domination* (Doble dominación). Similar al modo *Capture the Flag*, este modo por equipos consiste en controlar de manera presencial y simultánea, la base del propio equipo y del contrario durante un tiempo determinado.

- *Bombing Run* (Carrera de bombardeo). Modalidad por equipos con un esquema similar al de un partido de fútbol adaptado a los escenarios de UT04. Gana el equipo que consiga pasar una bola por un portal un número determinado de veces en un tiempo prefijado.

- *Last Man Standing* (Último hombre en pie). Similar al modo *Deathmatch*, pero en este caso cada jugador participa con un número limitado de vidas en la partida.

- *Invasion* (Invasión). Juego de equipo en el que los jugadores deben combatir juntos contra las oleadas de monstruos invasores que aparecerán puntualmente en el escenario.

- *Mutant* (Mutante). Modalidad con carácter mixto en el que el primer personaje que consiga matar a un rival, se convertirá en un ser mutante. Desde ese momento todos los demás intentarán eliminarlo (el jugador que lo consiga pasará a ser el mutante, en detrimento del anterior). Gana el que más veces mate al mutante al final de la ronda.

- *Onslaught* (Avalancha). Similar al modo *Double Domination*, en esta ocasión los puntos a controlar conforman caminos que conectan con la base de cada equipo. El primer equipo que consiga controlar todos los nodos del camino a la base enemiga, podrá destruirla, anotándose un punto.

En cuanto a la elección de la modalidad a utilizar durante la fase de experimentación, ésta deberá ser de las correspondientes a modos individuales, puesto que el resto de modos de juego requieren de un nivel cognitivo mayor del que se plantea alcanzar en este estudio, al tener que desarrollar tácticas de equipo que, en ocasiones, pueden resultar de extrema complejidad de cara a solventar los objetivos marcados.

Existen por tanto dos alternativas: *Deathmatch* y *Last Man Standing*, de entre las que se escogerá la primera, por permitir un mayor margen de maniobra a la hora de realizar experimentos, ya que no implica necesariamente un límite de muertes a alcanzar.

Herramientas utilizadas

A lo largo de esta sección, se describirán aspectos importantes relativos al funcionamiento de las diversas herramientas utilizadas para realizar el estudio.

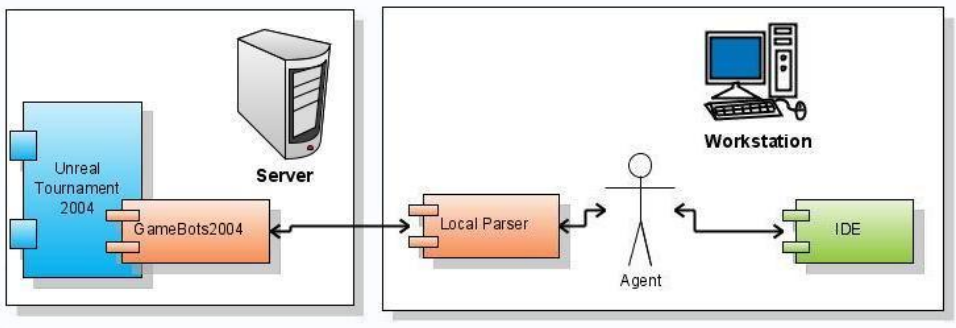

Figura 8 Arquitectura de Gamebots 2004

Pogamut

Pogamut es un proyecto que proporciona los mecanismos necesarios para realizar labores de prototipado y depuración de agentes autónomos virtuales a un cliente dentro de la plataforma UT04. Esta herramienta se integra con el entorno de programación NetBeans [*30*] como un plugin (ver *Glosario*) adicional

y permite definir el comportamiento de los agentes mediante el uso de dos lenguajes de programación: Java [*31*] y Python [*32*].

Como se puede deducir de la *Figura 8*, el proyecto Pogamut es el encargado de las siguientes tareas:

- Traducir la información procedente del servidor mediante el uso de un traductor (o *parser*) local. El módulo traductor es un elemento que precede al cliente y permite simplificar el procesamiento de los mensajes de Gamebots, convirtiéndolos de su formato ASCII (ver *Glosario*) original, a objetos propios del lenguaje Java. Además, el traductor se encarga de reducir el flujo de información que llega al cliente, puesto que sólo le envía aquella que ha sufrido modificaciones desde el último envío.

- Devolver al cliente toda la información del agente relativa a su propio estado o a la situación del entorno que lo rodea en cada momento de la partida. Esta gestión de información, queda reflejada en NetBeans, ya que dicha herramienta se encarga de mostrar los registros del agente obtenidos por Pogamut. La información que llega a dichos registros del agente, puede ser síncrona (toda aquella procedente de llamadas realizadas por el usuario) o asíncrona (la notificación de eventos ocurridos a lo largo de la partida en momentos puntuales en imprevisibles y que pueden requerir de un trato especial por parte del agente, si así está definido en su comportamiento).

Gamebots

Es el módulo encargado de la gestión de información en el lado del servidor. Gamebots es un mod de UT04 que permite controlar los distintos agentes autónomos virtuales mediante una interfaz de comunicación que utiliza mensajes de texto en formato ASCII para transmitir datos.

De este modo, no sólo se pueden controlar los agentes del juego mediante comandos de texto, sino que se recibe información relativa al estado de la partida en cada instante de tiempo.

4. Trabajo Realizado

A lo largo de esta sección, se detallarán los pasos seguidos para llevar a cabo cada una de las iteraciones a través del modelo en espiral de ciclo de vida del software. Cada ciclo estará compuesto a su vez de las cuatro fases explicadas en el apartado de *Gestión y Organización del Proyecto*, a través de las cuales se definirán diversos catálogos de requisitos, estudios de riesgos y fases de desarrollo y pruebas de prototipos. Del mismo modo, se proporcionarán diagramas cuando estos puedan ser de utilidad para comprender el funcionamiento de la solución propuesta.

En cuanto al análisis de requisitos de cada ciclo, dichos aspectos se recogerán en tablas que contendrán toda la información necesaria para su correcta comprensión. Los campos contenidos en las tablas son los siguientes:

- Identificador. Contendrá una cadena de caracteres, que identificará el tipo de requisito representado ("*RU*", en el caso de requisitos de usuario; "*RF*", en el caso de requisitos funcionales; "*RNF*", en caso de no funcionales), seguida de una cadena de dígitos que identifica de manera unívoca a cada uno de los requisitos y que será asignada de forma secuencial para cada tipo de requisito, comenzando en 001.

- Nombre. Título del requisito.

- Actores. Entidades que intervienen en la consecución del requisito.

- Descripción. Breve explicación del propósito del requisito.

- Origen. Determina la entidad concreta que origina el requisito.

- Verificable. Se especifica si puede asegurarse el cumplimiento del requisito. Tendrá valor "*Sí*", si se puede; y "*No*", en caso contrario.

- Claro. Determina si existe algún tipo de ambigüedad posible en la definición del requisito. Tendrá valor "*Sí*", si no hay ambigüedades; y "*No*", en caso contrario.

- Prioridad. Establece el grado de importancia que este requisito tiene para el modelo a desarrollar. Se mide con una escala numérica que va desde 1 (prioridad mínima) hasta 5 (prioridad máxima).

- Necesidad. Este campo puede tomar dos valores. "*Esencial*", si su aplicación es totalmente necesaria; o "*Negociable*", si es aconsejable aplicarlo, pero no obligatorio.

- Estabilidad. Rango de valores numéricos desde 1 (menos estable) hasta 5 (más estable), que determina si el requisito está sujeto a cambios frecuentes.

Requisitos de Usuario

Los requisitos de usuario, constituyen una referencia común para todos los ciclos a desarrollar a lo largo del ciclo de vida. Por ello, se detallan a continuación:

Identificador	RU001				
Nombre	Enfoque General				
Actores	Sistema				
Descripción	El sistema debe proporcionar un prototipo de agente autónomo capaz de participar en partidas de videojuegos FPS en modo multijugador.				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RU 1 Enfoque general del sistema

Identificador	RU002				
Nombre	Uso de Inteligencia Artificial				
Actores	Sistema				
Descripción	El sistema debe aplicar alguna técnica de Inteligencia Artificial al comportamiento del bot a desarrollar.				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RU 2 Aplicación de IA en el sistema

Identificador	RU003				
Nombre	Limitaciones de Tiempo				
Actores	Sistema				
Descripción	El sistema desarrollará partidas de duración predefinida y responderá a los estímulos del entorno en tiempo real				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	3	**Necesidad**	Negociable	**Estabilidad**	4

RU 3 Restricciones de tiempo del sistema

Identificador	RU004				
Nombre	Protocolo de Comunicación				
Actores	Sistema				
Descripción	El sistema debe aportar un protocolo de comunicación entre el servidor de UT04 y el cliente correspondiente a cada uno de los agentes autónomos implicados en la partida.				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	4	**Necesidad**	Esencial	**Estabilidad**	5

RU 4 Protocolo de comunicación del sistema

Identificador	RU005				
Nombre	Adaptabilidad del Sistema				
Actores	Sistema				
Descripción	El sistema debe ser fácilmente adaptable a los cambios en las especificaciones de requisitos				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	4	**Necesidad**	Esencial	**Estabilidad**	5

RU 5 Capacidad de adaptación a cambios

Identificador	RU006				
Nombre	Portabilidad del Sistema				
Actores	Sistema				
Descripción	El sistema debe ser portable entre diversos sistemas operativos				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	3	**Necesidad**	Negociable	**Estabilidad**	4

RU 6 Posibilidad de portar el sistema

Identificador	RU007				
Nombre	Robustez del Sistema				
Actores	Sistema				
Descripción	El sistema debe proporcionar mecanismos de control de excepciones en tiempo de ejecución				
Origen	Usuario				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RU 7 Robustez del sistema a implementar

Ciclo 1: Desarrollo mediante Sistemas Expertos

En esta primera iteración del ciclo, se busca un acercamiento al prototipado de agentes inteligentes aplicables a UT04 mediante el uso de sistemas expertos. Estos sistemas constituyen una técnica de Inteligencia Artificial basada en el uso de una serie de reglas recogidas por un experto en la materia y almacenadas en una base de conocimiento.

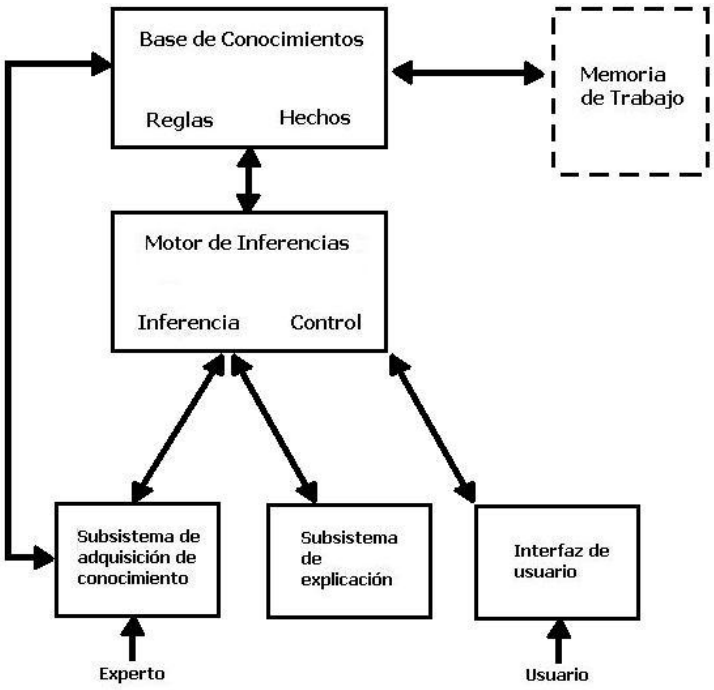

Figura 9 Estructura de sistema experto

Como se puede observar en el esquema de la *Figura 9*, la definición formal de un sistema experto implica contar con una base de conocimientos, donde se almacena toda la experiencia de los profesionales en la materia; un motor de inferencias, el cual modela el razonamiento humano a utilizar durante la ejecución del sistema y una serie de módulos utilizados para incorporar nuevos conocimientos al sistema, o recuperar los ya existentes.

Sin embargo, a lo largo de este primer ciclo, se llevará a cabo una definición informal del sistema experto, basada en reglas. Es decir, no existirá una implementación estricta del modelo de la figura anterior, sino que se verá representado mediante estructuras condicionales que permitan realizar una acción determinada ante determinadas situaciones de juego.

Con el desarrollo de esta primera versión del código, se pretende alcanzar un nivel de complejidad suficiente, que permite al agente reaccionar ante determinados eventos acontecidos en el escenario. El siguiente diagrama de casos de uso muestra todas las acciones que el bot puede realizar basándose en las reglas que definen su comportamiento:

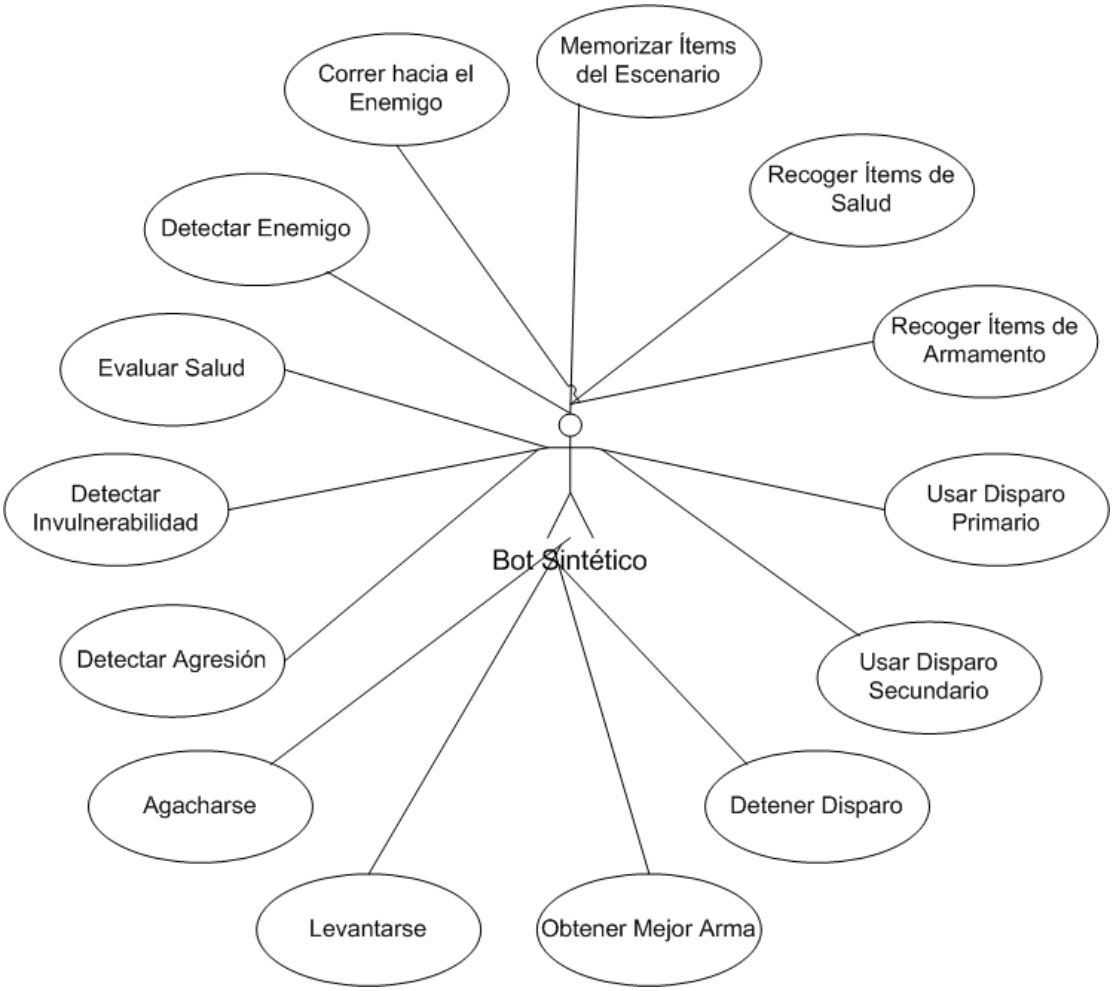

Figura 10 Diagrama de casos de uso del agente basado en sistemas expertos

A continuación se detallarán cada una de las acciones a realizar, especificando en cada caso, una descripción del caso de uso, las precondiciones, flujo y postcondiciones existentes, así como (si los hubiere) flujos alternativos ante algún tipo de inconveniente al realizar el flujo básico:

Nombre	Detectar Enemigo
Descripción	Comprueba si existe algún enemigo dentro del campo visual del bot
Flujo Básico	El bot comprueba si hay algún enemigo
Postcondiciones	1. Si hay uno a. Obtiene sus datos 2. Si hay varios a. Obtiene los datos de uno de ellos
Flujo Alternativo	Si no hay enemigos en el campo de visión del agente, éste obtendrá un valor nulo en respuesta a su consulta

CU 1 Detección de un enemigo

Nombre	Evaluar Salud
Descripción	Comprueba el nivel actual de salud del bot
Flujo Básico	El bot solicita estado actual de salud
Postcondiciones	Obtiene datos relativos a su nivel de salud

CU 2 Evaluación de salud del agente

Nombre	Detectar Invulnerabilidad
Descripción	Comprueba si el enemigo es vulnerable a armas de fuego
Precondiciones	El enemigo debe haberse detectado previamente
Flujo Básico	1. Detecta enemigo 2. Solicita información sobre invulnerabilidad del enemigo
Postcondiciones	Obtiene información relativa a la invulnerabilidad del enemigo

CU 3 Detección de invulnerabilidad de un jugador

Nombre	Detectar Agresión
Descripción	El bot comprueba si está recibiendo un ataque procedente de algún enemigo
Flujo Básico	Solicita información sobre su estado interno
Postcondiciones	Obtiene los datos solicitados

CU 4 Detección de una agresión

Nombre	Obtener mejor arma
Descripción	Descubre cuál de las armas de su inventario es más apropiada para abordar un ataque en una distancia especificada
Precondiciones	Deben conocerse dos puntos del espacio (origen y destino del proyectil)
Flujo Básico	Solicita información sobre el arma más recomendable
Postcondiciones	Recibe el identificador del arma en cuestión

CU 5 Obtención de la mejor arma

Nombre	Usar Disparo Primario
Descripción	Se encarga de utilizar el disparo primario del arma actual
Precondiciones	1. La munición primaria del arma debe ser mayor que cero 2. Se debe conocer la ubicación de un enemigo
Flujo Básico	El bot abre fuego primario contra un enemigo concreto
Postcondiciones	1. El arma se dispara 2. La cantidad de munición primaria del arma disminuye

CU 6 Uso del disparo principal

Nombre	Usar Disparo Secundario
Descripción	Se encarga de utilizar el disparo alternativo del arma actual
Precondiciones	1. La munición secundaria del arma debe ser mayor que cero 2. Se debe conocer la ubicación de un enemigo
Flujo Básico	El bot abre fuego alternativo contra un enemigo concreto
Postcondiciones	1. El arma se dispara 2. La cantidad de munición secundaria del arma disminuye

CU 7 Uso del disparo alternativo

Nombre	Detener Disparo
Descripción	El bot para de disparar su arma
Precondiciones	El bot debe estar disparando el arma previamente
Flujo Básico	El bot solicita dejar de disparar
Postcondiciones	El fuego se detiene

CU 8 Detención de ataque

Nombre	Agacharse
Descripción	El bot flexiona las rodillas
Precondiciones	El agente debe estar previamente situado en una posición erguida.
Flujo Básico	El bot solicita agacharse
Postcondiciones	El bot se agacha

CU 9 Pasar a posición agachada

Nombre	Levantarse
Descripción	El bot pasa a una posición erguida
Precondiciones	El bot debe estar previamente agachado
Flujo Básico	Solicita levantarse
Postcondiciones	El bot se levanta

CU 10 Pasar a posición erguida

Nombre	Correr hacia el Enemigo
Descripción	Consiste en aproximarse al rival con el fin de ser más preciso al atacarle
Precondiciones	El bot debe haber detectado previamente la posición del enemigo
Flujo Básico	1. Detecta al enemigo 2. Decide ir hacia él
Postcondiciones	El bot persigue constantemente la última posición conocida del enemigo

CU 11 Perseguir al enemigo

Nombre	Memorizar Ítems del Escenario
Descripción	Consiste en recopilar información del servidor al comenzar la partida acerca de los puntos en los que se encuentran diversos ítems como armas, salud, etc
Flujo Básico	El bot solicita información relativa a los puntos del espacio en los que se encuentran diversos elementos
Postcondiciones	Se reciben listado de las posiciones de dichos ítems, organizados por su función
Flujo Alternativo	Todos los mapas no poseen los mismos ítems, lo que implica que al solicitar alguno de ellos, no se encuentren disponibles en el mapa actual. En ese caso, se obtiene un valor nulo.

CU 12 Memorización de elementos del escenario

Nombre	Recoger Ítems de Salud
Descripción	El bot pasa por los puntos del escenario en los que se encuentran bonificaciones de salud
Precondiciones	El agente debe conocer previamente los puntos del escenario en que se encuentran los ítems de salud
Flujo Básico	El agente decide recorrer el grafo conexo que une todos los ítems de salud a través del mapa
Postcondiciones	El agente adquiere una bonificación de salud que dependerá del ítem recogido y del nivel previo de vida
Flujo Alternativo	1. En ningún caso puede superarse el umbral máximo de vida (inicialmente, el valor por defecto son 100 puntos y el máximo son 199). Si se sobrepasase este límite al coger un ítem, el nivel será fijado al máximo estipulado 2. Si breves instantes antes de pasar por un punto en el que debe haber un ítem de salud, algún jugador ha cogido la bonificación, está no estará disponible hasta pasados unos segundos

CU 13 Recogida de ítems de salud

Nombre	Recoger Ítems de Armamento
Descripción	El bot pasa por los puntos del escenario en los que hay armas
Precondiciones	El agente debe conocer previamente los puntos del escenario en que se encuentran las armas
Flujo Básico	El agente decide recorrer el grafo conexo que une todos los ítems a través del mapa
Postcondiciones	El agente adquiere un nuevo arma o munición de la misma (si ya poseía dicho ítem con anterioridad)
Flujo Alternativo	1. En ningún caso puede superarse el umbral máximo de vida (inicialmente, el valor por defecto son 100 puntos y el máximo son 199). Si se sobrepasase este límite al coger un ítem, el nivel será fijado al máximo estipulado. 2. Si breves instantes antes de pasar por un punto en el que debe haber un ítem de salud, algún jugador ha cogido la bonificación, está no estará disponible hasta pasados unos segundos.

CU 14 Recogida de ítems de armamento

Análisis de Requisitos

En esta primera fase del ciclo de vida, se llevará a cabo un estudio de la situación que se pretende alcanzar mediante el desarrollo de la solución. A continuación se muestran los requisitos de software correspondientes a este primer prototipo que complementan a los requisitos de usuario vistos con anterioridad en la sección *Requisitos de Usuario*:

Identificador	RF001				
Nombre	Uso de Funcionalidad				
Actores	Agente				
Descripción	El agente debe ser capaz de utilizar toda la funcionalidad que le proporcione la interfaz de Gamebots				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 1 Uso de la funcionalidad proporcionada por Gamebots

Identificador	RF002				
Nombre	Videojuego Utilizado				
Actores	Sistema				
Descripción	El sistema debe desarrollarse orientado a la plataforma de juegos Unreal Tournament 2004				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 2 Plataforma de videojuegos a utilizar

Identificador	RF003				
Nombre	Interfaz de Comunicación				
Actores	Sistema				
Descripción	El sistema debe hacer uso de la interfaz de comunicación proporcionada por el mod Gamebots				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 3 Interfaz de comunicación a utilizar

Identificador	RF004				
Nombre	Lenguaje Utilizado				
Actores	Sistema				
Descripción	El sistema debe modelarse mediante el lenguaje Java				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 4 Lenguaje de programación a utilizar

Identificador	RF005				
Nombre	Proyecto Generado				
Actores	Sistema				
Descripción	El sistema debe integrarse en un proyecto Pogamut				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 5 Integración del código en un proyecto

Identificador	RF006				
Nombre	Entorno de Desarrollo				
Actores	Sistema				
Descripción	El sistema debe ser implementado mediante el entorno de desarrollo NetBeans				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 6 Entorno de programación a utilizar

Identificador	RF007				
Nombre	Nivel de Cognición				
Actores	Agente				
Descripción	El agente debe alcanzar una capacidad cognitiva que satisfaga al menos, el nivel reactivo de la escala ConsScale				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 7 Nivel cognitivo mínimo a alcanzar

Identificador	RF008				
Nombre	Técnica de IA Aplicada				
Actores	Agente				
Descripción	El sistema aportará conceptos procedentes de sistemas expertos para dotar al agente de cierto grado de inteligencia				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 8 Técnica de IA a aplicar en el sistema

Identificador	RNF001				
Nombre	Duración de las Partidas				
Actores	Gamebots				
Descripción	Gamebots debe proporcionar escenarios de juego limitados a 2 horas de juego o un límite de 25 muertes que deberá alcanzar un jugador para ganar la ronda				
Origen	RU003				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 1 Restricción de duración de la partida

Identificador	RNF002				
Nombre	Protocolo de Comunicación				
Actores	Agente, Gamebots				
Descripción	La interfaz de comunicación existente entre el agente autónomo y el servidor de Gamebots utilizará un protocolo TCP/IP para intercambiar mensajes de texto en claro				
Origen	RU004				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 2 Protocolo de comunicación entre el bot y Gamebots

Identificador	RNF003				
Nombre	Muerte de un Rival				
Actores	Gamebots, agente				
Descripción	El sistema incrementará el marcador del bot en una unidad cuando éste elimine a otro jugador durante el desarrollo de una partida				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 3 Muerte de un rival

Identificador	RNF004				
Nombre	Muerte del Agente				
Actores	Gamebots, agente				
Descripción	El sistema mantendrá igual el marcador del bot cuando éste sea eliminado por otro jugador durante el desarrollo de una partida				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 4 Muerte del propio agente

Identificador	RNF005				
Nombre	Suicidio de un Rival				
Actores	Gamebots, agente				
Descripción	El sistema decrementará el marcador del bot en una unidad cuando éste se elimine a sí mismo durante el desarrollo de una partida				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 5 Suicidio del agente

Identificador	RNF006				
Nombre	Versión de NetBeans				
Actores	Sistema				
Descripción	La versión a emplear de NetBeans será la 6.1				
Origen	RU001, RF006				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 6 Versión de NetBeans a utilizar

Identificador	RNF007				
Nombre	Versión de Pogamut				
Actores	Sistema				
Descripción	La versión a emplear de Pogamut será la 2.3.3				
Origen	RU001, RF005				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 7 Versión de Pogamut a utilizar

Identificador	RNF008				
Nombre	Versión de Gamebots				
Actores	Sistema				
Descripción	La versión a emplear de Gamebots será la 2004				
Origen	RU001, RF003				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 8 Versión de Gamebots a utilizar

Identificador	RNF009				
Nombre	Versión de Java				
Actores	Sistema				
Descripción	La versión del kit de desarrollo de Java (JDK) será la 6				
Origen	RU001, RF004				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 9 Versión de Java a utilizar

Identificador	RNF010				
Nombre	Requisitos de Software				
Actores	Sistema				
Descripción	Los requisitos mínimos del sistema en el que se instale UT04 deben ser: • Procesador Pentium III o AMD Athlon 1.0 GHz • 128 MB de memoria RAM • 5.5 GB de espacio disponible en disco • CD-ROM o DVD de 8x • Tarjeta de sonido compatible con Windows • Tarjeta gráfica de 32 MB • Version 9.0b de DirectX • Juego en red local (TCP/IP) soportado				
Origen	RU005				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 10 Requisitos mínimos de Unreal Tournament 2004

Identificador	RNF011				
Nombre	Sistemas Operativos Compatibles				
Actores	Sistema				
Descripción	El sistema debe ser ejecutable desde Windows y Linux				
Origen	RU006				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 11 Compatibilidad de sistemas operativos

Identificador	RNF012				
Nombre	Calidad del Prototipo				
Actores	Sistema				
Descripción	El sistema debe proporcionar un conjunto de pruebas de ejecución en las que se pueda garantizar la eficacia del agente autónomo desarrollado				
Origen	RU001				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 12 Garantía de calidad del prototipo implementado

Identificador	RNF013				
Nombre	Control de Excepciones				
Actores	Sistema				
Descripción	El sistema aportará estructuras de tratamiento de excepciones para evitar que su presencia detenga la ejecución del comportamiento del bot				
Origen	RU007				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 13 Control de excepciones durante la ejecución del sistema

Identificador	RNF014				
Nombre	Especificaciones sobre Técnica Aplicada				
Actores	Sistema				
Descripción	El sistema experto debe utilizar sentencias condicionales para determinar en cada momento la acción a determinar				
Origen	RU002, RF008				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 14 Especificaciones sobre la técnica de IA empleada

Finalmente, se muestra una matriz de trazabilidad, cuyo objetivo es comprobar que todos los requisitos de usuario tienen, al menos una

correspondencia en el conjunto de requisitos del software que detallan cómo se debe llevar a cabo el cumplimiento de las especificaciones:

	RU001	RU002	RU003	RU004	RU005	RU006	RU007
RF001	X						
RF002	X						
RF003	X						
RF004	X						
RF005	X						
RF006	X						
RF007		X					
RF008		X					
RNF001			X				
RNF002				X			
RNF003	X						
RNF004	X						
RNF005	X						
RNF006	X						
RNF007	X						
RNF008	X						
RNF009	X						
RNF010					X		
RNF011						X	
RNF012	X						
RNF013							X
RNF014		X					

Tabla 2 Matriz de trazabilidad del sistema

Riesgos, Dificultades y Oportunidades de la Arquitectura

Como se ha podido observar en la fase de *Análisis de Requisitos* el objetivo en esta primera iteración del modelo de ciclo de vida en espiral consistirá en realizar un prototipo de agente sintético que base su comportamiento en una serie de reglas procedentes de un sistema experto. Para el desarrollo de dicho sistema, se utilizarán las herramientas Unreal Tournament 2004, Gamebots 2004, Pogamut 2 y NetBeans 6.1. A lo largo de este apartado, se llevará a cabo un estudio de los riesgos, ventajas e inconvenientes que presenta esta alternativa.

En cuanto a los riesgos presentes en cualquier implementación que cumpla con los requisitos anteriormente descritos en las secciones de *Requisitos de Usuario* y *Análisis de Requisitos*, los principales problemas que se pueden encontrar son:

- Mala elección de las acciones a emplear para modelar el comportamiento del bot. El uso de acciones innecesarias o, por el contrario, la ausencia de acciones prioritarias, pueden conllevar un comportamiento extremadamente complejo (o simple) que reporte pocos beneficios al rendimiento del agente.

- Mala definición de las condiciones. Si las situaciones que implican el uso de determinadas acciones están mal definidas, no servirá de nada haber aplicado correctamente el conocimiento a la hora de elegir dichas actividades a realizar, puesto que cada una de ellas es recomendable en situaciones muy puntuales de la ejecución, y su uso normalmente no es generalizable a otro tipo de eventos (por ejemplo, disparar es bueno si vemos un enemigo delante, pero no si no hay nadie, ya que en ese caso podemos quedarnos sin munición y estar indefensos ante una posterior aparición de agentes contrarios).

Ambas situaciones pueden ser fácilmente evitadas, contando en el sistema experto con una base del conocimiento de alta calidad, generada a partir de las opiniones de diversas personas experimentadas en el uso de UT04.

En cuanto a las dificultades que presenta esta alternativa, algunas de las más importantes son las siguientes:

- Comportamiento rígido del bot. Este aspecto plantea un inconveniente importante a la hora de tratar con situaciones inicialmente no contempladas por las reglas del sistema experto, ya que el agente autónomo no será capaz de tratarlas convenientemente.

- Incapacidad para aprender. Este tipo de técnicas de IA es incapaz de realizar un aprendizaje en tiempo real. Es decir, sólo puede evolucionar si se le añaden manualmente nuevas reglas que serían aplicadas desde la siguiente ejecución. Este factor redunda en la rigidez del comportamiento del agente, al no poder incorporar una realimentación automática ante eventos acontecidos en instantes anteriores de la partida.

Finalmente, esta solución también presenta ciertas ventajas respecto a otras técnicas de IA aplicables a este proyecto:

- Mayor capacidad cognitiva inicial. Mientras que con otras técnicas es preciso llevar a cabo un proceso de aprendizaje o entrenamiento para poder alcanzar niveles de rendimiento aceptables, los sistemas expertos permiten contar con los conocimientos heredados de profesionales de la materia desde el instante inicial, con lo que su comportamiento resulta más heterogéneo a lo largo de toda la ejecución.

- Robustez ante el ruido. Durante el proceso de entrenamiento del agente con otras técnicas de IA, se puede experimentar cierto ruido procedente de comportamientos anómalos de otros jugadores, que implicarían un aprendizaje de reglas de comportamiento deficientes. Al aplicarse sistemas expertos, este ruido queda minimizado.

Tras analizar todas las ventajas e inconvenientes que presenta la opción de desarrollar un prototipo de agente autónomo mediante sistemas expertos, se llega a la conclusión de que resulta una opción viable.

Implementación de Prototipo

La implementación de un prototipo de agente inteligente mediante sistemas expertos, supone una solución centrada en un sistema basado en reglas. El comportamiento del bot estará formado por una serie de reglas almacenadas, procedentes del conocimiento de personal experimentado en el juego. Por tanto, en cada momento el bot "sabe" que, si se da un evento determinado, ha de reaccionar con un comportamiento predefinido y constante.

A continuación se muestra un diagrama de actividad correspondiente al método *doLogic()* de la clase principal del proyecto Pogamut desarrollado (dicho método es llamado de forma indefinida durante toda la ejecución del agente, por lo que el diagrama sería aplicado de forma constante durante el tiempo de juego):

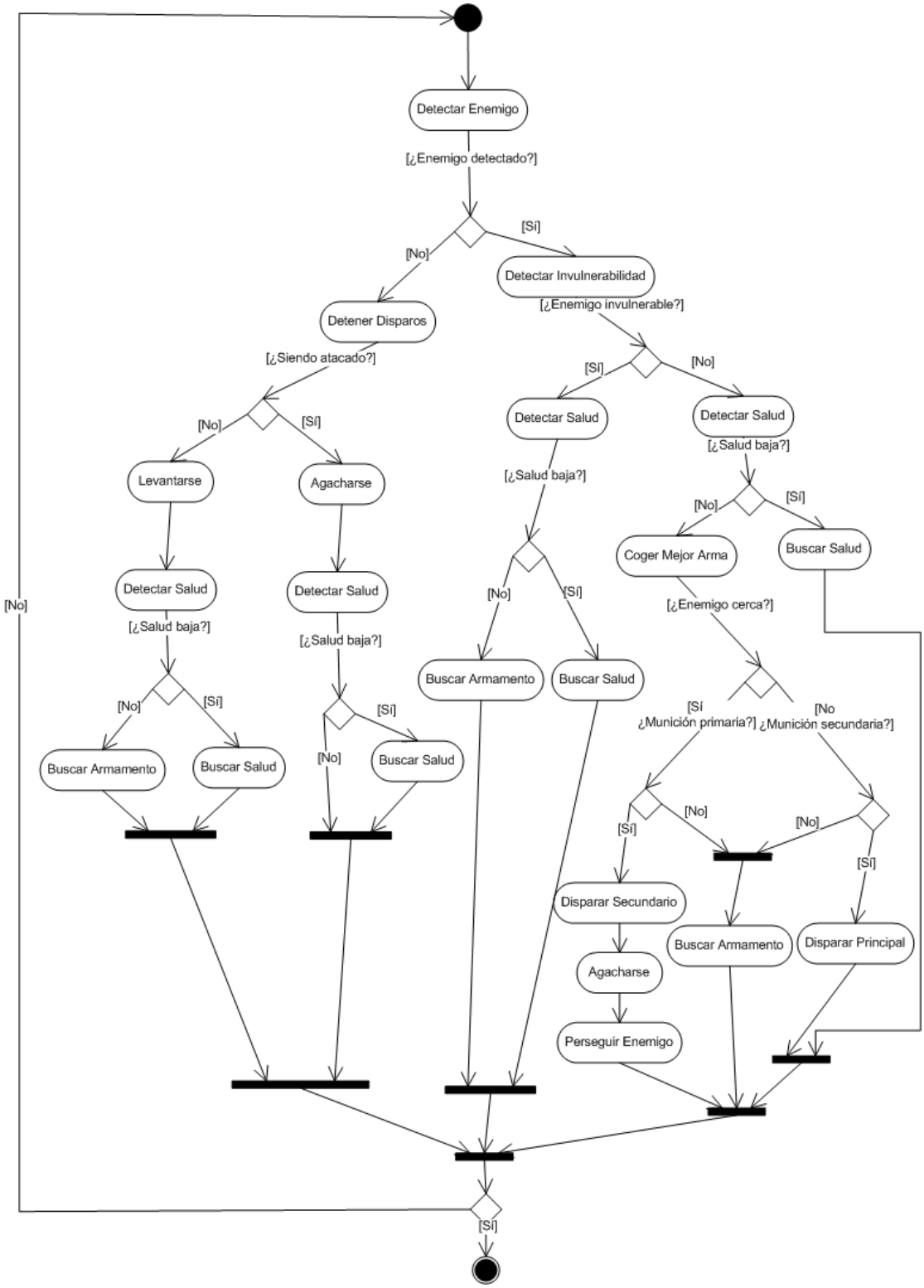

Figura 11 Diagrama de actividad del agente basado en sistemas expertos

Como se puede apreciar en la *Figura 11*, el comportamiento seguido por el bot sigue patrones lógicos de comportamiento ante determinados eventos del juego. De este modo, se comprobará:

- Conveniencia de ataque ante la presencia de un rival. Factores como la invulnerabilidad del enemigo o niveles bajos de munición o salud pueden afectar a la decisión final.

- La distancia del enemigo. Se utiliza para decidir el tipo de arma y el modo de disparo a utilizar (si fuera preciso).

- Necesidad de buscar ítems de salud o armamento de cara a evitar situaciones de peligro para el bot. Dichas acciones también serán realizadas en aquellos casos en que no se dé ningún evento prioritario, como una acción ofensiva o defensiva.

Una vez realizado este primer prototipo, resulta obvio que éste no cuenta con una gran cantidad de acciones disponibles en su haber, pero este factor lo convierte en una alternativa que, aunque simple, satisface todas las necesidades básicas de ataque y defensa de un agente autónomo.

En cuanto a la estructura de código generada, el diagrama de clases es el siguiente:

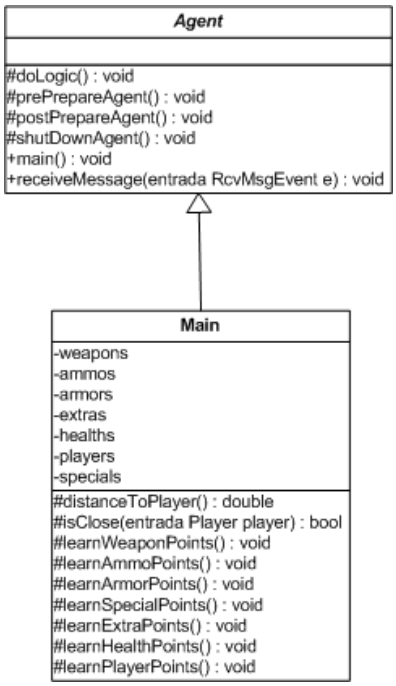

Figura 12 Diagrama de clases de agente basado en sistemas expertos

Como se puede observar en la *Figura 12*, existe una única clase dentro del código del proyecto Pogamut llamada *Main*, que hereda de una clase del API (ver *Glosario*) de Pogamut llamada *Agent*, la cual contiene toda la funcionalidad básica para compilar y ejecutar de un jugador. Dicha clase será ejecutada al utilizar al bot en una partida. Además de los métodos utilizados para obtener información en tiempo de ejecución acerca del estado del agente y sus niveles, existen cuatro métodos importantes:

- doLogic(). Método necesario para la ejecución de la clase, que es llamado constantemente durante el periodo en que el bot participa en la partida. Contiene toda la funcionalidad que el agente debe reproducir a lo largo del juego y, por tanto aplica las reglas pertenecientes al sistema experto.

- prePrepareAgent(). Método necesario para la ejecución de la clase, que contiene inicializaciones de atributos de la clase. Es llamado una única vez antes de comenzar a jugar.

- postPrepareAgent(). Método necesario para la ejecución de la clase, que contiene inicializaciones de parámetros relativos a elementos de comunicación del agente. Se llama una vez antes de comenzar la partida y, en este caso, se encarga de memorizar los puntos en los que se encuentran todos los ítems del escenario.

- shutDownAgent(). Se encarga de realizar actividades al finalizar la ejecución del agente. En este caso, se limita a mostrar una traza sobre el número de muertes que realizó el bot durante la partida.

Evaluación del Prototipo

Dado que este prototipo es el primero de cuantos se llevarán a cabo durante el ciclo de vida del software, no es posible realizar pruebas de carácter comparativo con otras alternativas del modelo. Sin embargo y, hasta que se pueda desarrollar una batería final de pruebas entre todos los prototipos, se puede probar la capacidad del agente, enfrentándolo en una partida de tipo Deathmatch contra un bot semejante. Es decir, se realizará una partida en la que se enfrenten dos agentes desarrollados mediante la misma técnica de sistemas expertos.

La hipótesis inicial es, más allá de vaticinar que uno de los dos agentes implementados mediante sistemas expertos ganará la partida, que dicho desenlace se conseguirá en un tiempo muy inferior a las dos horas de tiempo máximo estipuladas para una partida. Además, el resultado alcanzado por ambos agentes deberá ser muy similar, ya que su comportamiento es idéntico y estadísticamente el rendimiento de ambos debería de ser de un 50%.

Una vez obtenidos los resultados, comprobamos los siguientes datos:

Tiempo Transcurrido:	22 minutos
Marcador Agente Ganador:	25 muertes
Marcador Agente Perdedor:	22 muertes
Tasa de Igualdad:	1,09

Tabla 3 Evaluación de prototipo experto

La tasa de igualdad es un cálculo que refleja cuán similares son ambos bots en materia de rendimiento. Es calculada de la siguiente manera:

$$Tasa = \frac{Marcador_{ganador} - 1}{Marcador_{perdedor}}$$

Tasas próximas a uno, denotarán una gran igualdad en términos de eficacia entre ambos agentes. Por otro lado, tasas muy superiores a uno, implicarán una clara desigualdad en el comportamiento de ambos agentes.

Como ya se comentó al realizar la hipótesis, los resultados reflejan la gran similitud entre el rendimiento de ambos agentes, ya que se rigen por el mismo patrón de comportamiento. Respecto al tiempo transcurrido, es preciso decir que denota un patrón netamente agresivo de los agentes, puesto que han logrado finalizar la partida en una sexta parte del tiempo total estipulado. Este aspecto debe ser valorado en función del rival a batir, puesto que es una estrategia muy aceptable al enfrentarse a un agente que tienda a realizar principalmente labores defensivas, pero no tanto en casos como el actual en los que el contrincante también es de carácter violento (ya que si su nivel de complejidad es superior al de nuestro agente, incurriremos en resultados muy negativos).

El comportamiento de un rival no puede conocerse de antemano, por lo que sería altamente recomendable realizar una nueva iteración del modelo en espiral, con el fin de generar un agente capaz de amoldarse a diversos comportamientos de manera dinámica. Esto implica la utilización de mecanismos de aprendizaje, que permitan al agente tener en cuenta situaciones vividas con anterioridad y adaptarse a ellas. Por ello, el próximo agente contará con dicho mecanismo.

Ciclo 2: Desarrollo mediante Aprendizaje por Refuerzo

En esta segunda etapa del ciclo de vida, se pretende producir un prototipo del agente autónomo que mejore determinadas carencias observadas durante la fase anterior. Para ello, se ha considerado un conjunto de acciones mayor que en el caso anterior, con el fin de dotar al bot de un abanico de posibilidades que le permita reaccionar y adaptarse a las situaciones del juego de forma más compleja e inteligente. Las acciones que aporta este diseño de forma adicional al diagrama del anterior modelo (ver *Figura 10*) son las siguientes:

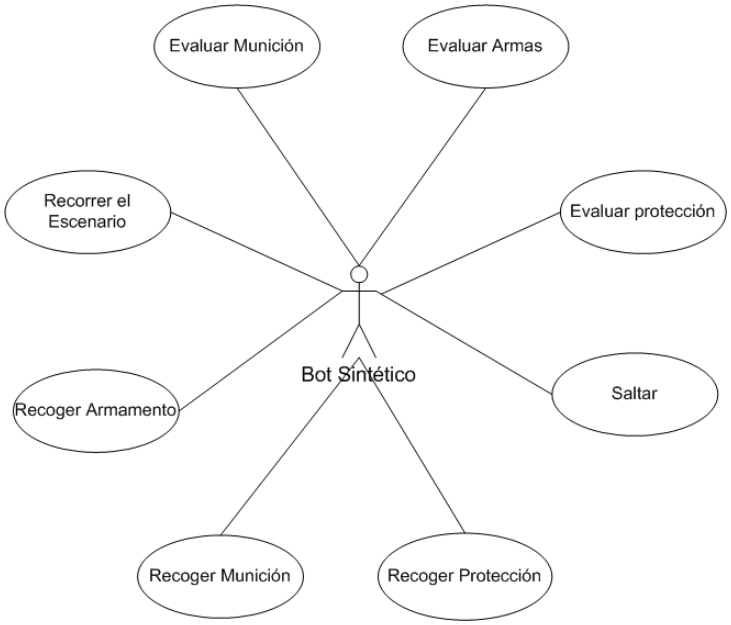

Figura 13 Diagrama de casos de uso del agente con aprendizaje por refuerzo

Nombre	Evaluar Munición
Descripción	Comprueba el nivel de munición del arma actualmente utilizada por el bot
Flujo Básico	El bot solicita estado actual de munición
Postcondiciones	Obtiene datos relativos a su nivel de munición

CU 15 Evaluación de la munición restante del agente

Nombre	Evaluar Armas
Descripción	Comprueba el número actual de armas que hay en el inventario del bot
Flujo Básico	El bot solicita el número actual de armas distintas que posee
Postcondiciones	Obtiene datos relativos a su número de armas

CU 16 Evaluación de las armas que posee el bot

Nombre	Evaluar Protección
Descripción	Comprueba el nivel actual de protección del bot
Flujo Básico	El bot solicita estado actual de su armadura
Postcondiciones	Obtiene datos relativos a su nivel de armadura

CU 17 Evaluación de la protección con que cuenta el agente

Nombre	Saltar
Descripción	El bot realiza saltos de forma reiterativa
Flujo Básico	Solicita saltar
Postcondiciones	El bot salta

CU 18 Acción de saltar

Nombre	Recorrer el Escenario
Descripción	El bot se dedica a recorrer el mapa pasando en cada momento por el punto de navegación (nodos del grafo que conforma el entramado de caminos que puede recorrer un bot en un mapa concreto) más cercano
Precondiciones	El bot debe haber detectado previamente un punto de navegación
Flujo Básico	Solicita llegar hasta el punto de navegación observado
Postcondiciones	El bot recorre la distancia existente entre dos puntos de navegación

CU 19 Circuito a través del escenario

Nombre	Recoger Ítems de Protección
Descripción	El bot pasa por los puntos del escenario en los que se encuentran bonificaciones de armadura
Precondiciones	El agente debe conocer previamente los puntos del escenario en que se encuentran los ítems de protección
Flujo Básico	El agente decide recorrer el grafo conexo que une todos los ítems de protección a través del mapa
Postcondiciones	El agente adquiere una bonificación de armadura que dependerá del ítem recogido y del nivel previo de armadura
Flujo Alternativo	1. En ningún caso puede superarse el umbral máximo de protección (el valor por defecto es 0 y el máximo es 150). Si se sobrepasase este límite al coger un ítem, el nivel será fijado al máximo estipulado 2. Si breves instantes antes de pasar por un punto en el que debe haber un ítem de protección, algún jugador ha cogido la bonificación, está no estará disponible hasta pasados unos segundos

CU 20 Búsqueda de elementos de protección

Nombre	Recoger Ítems de Munición
Descripción	El bot pasa por los puntos del escenario en los que se encuentran bonificaciones de munición
Precondiciones	El agente debe conocer previamente los puntos del escenario en que se encuentran los ítems de munición
Flujo Básico	El agente decide recorrer el grafo conexo que une todos los ítems de munición a través del mapa
Postcondiciones	El agente adquiere una bonificación de munición de un arma concreta, que dependerá del ítem recogido y del nivel previo de munición
Flujo Alternativo	1. En ningún caso puede superarse el umbral máximo de munición (varía en función del tipo de munición recogida). Si se sobrepasase este límite al coger un ítem, el nivel será fijado al máximo estipulado 2. Si breves instantes antes de pasar por un punto en el que debe haber un ítem de munición, algún jugador ha cogido la bonificación, está no estará disponible hasta pasados unos segundos

CU 21 Búsqueda de munición por el mapa

Nombre	Recoger Ítems de Armamento
Descripción	El bot pasa por los puntos del escenario en los que hay armas
Precondiciones	El agente debe conocer previamente los puntos del escenario en que se encuentran las armas
Flujo Básico	El agente decide recorrer el grafo conexo que une todos los ítems a través del mapa
Postcondiciones	El agente adquiere un nuevo arma o munición de la misma (si ya poseía dicho ítem con anterioridad)
Flujo Alternativo	1. En ningún caso puede superarse el umbral máximo de vida (inicialmente, el valor por defecto son 100 puntos y el máximo son 199). Si se sobrepasase este límite al coger un ítem, el nivel será fijado al máximo estipulado. 2. Si breves instantes antes de pasar por un punto en el que debe haber un ítem de salud, algún jugador ha cogido la bonificación, está no estará disponible hasta pasados unos segundos.

CU 22 Búsqueda de armas por el escenario

Análisis de Requisitos

Los problemas estudiados durante la etapa anterior derivan de la incapacidad del bot para evolucionar de forma dinámica. Por tanto, será preciso llevar a cabo una modificación de los requisitos de software definidos para el prototipo anterior (ver *Análisis de Requisitos* del ciclo 1). Los requisitos sujetos a cambios serán los identificados como *RF* 7, *RF* 8 y *RNF* 14 y quedarán de la siguiente manera:

Identificador	RF007				
Nombre	Nivel de Cognición				
Actores	Agente				
Descripción	El agente debe alcanzar una capacidad cognitiva que satisfaga al menos, el nivel adaptativo de la escala ConsScale				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 9 Modificación del RF007 del ciclo 1

Identificador	RF008				
Nombre	Técnica de IA Aplicada				
Actores	Agente				
Descripción	El sistema aplicará la técnica de aprendizaje por refuerzo para dotar al agente de cierto grado de inteligencia				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 10 Modificación del RF008 del ciclo 1

Identificador	RNF014				
Nombre	Especificaciones sobre Técnica Aplicada				
Actores	Sistema				
Descripción	El algoritmo de aprendizaje por refuerzo a utilizar será Q-Learning				
Origen	RU002, RF008				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 15 Modificación del RNF014 del ciclo 1

El nivel adaptativo de la escala ConsScale implica que el bot debe ser capaz de utilizar los sensores propioceptivos con la finalidad de alcanzar nuevas respuestas reactivas-adaptativas a los estímulos de su entorno. Es decir, la salida del sistema (las acciones a realizar) dependerá, no sólo de las entradas directas (estímulos del entorno), sino también de las salidas anteriores del sistema (resultado de situaciones similares realizadas en el pasado). Con ello se obtiene una realimentación del sistema que implica que el agente se adapte de forma dinámica a una situación determinada.

En cuanto al uso de Q-Learning para llevar a cabo la evolución del agente, se analizará su conveniencia en el siguiente apartado.

Riesgos, Dificultades y Oportunidades de la Arquitectura

El objetivo principal de este apartado será valorar la conveniencia de usar Q-Learning en el desarrollo del nuevo prototipo. En cuanto a los inconvenientes que plantea esta técnica, los más importantes son los siguientes:

- Problemas derivados del exceso de memoria requerido. A la hora de utilizar Q-Learning, es preciso diseñar una tabla en la que se almacenen todos los valores Q, que corresponderán a cada una de las relaciones estado-acción que se diseñarán en esta solución. Si el número de posibles combinaciones es demasiado grande, se corre el riesgo de requerir más memoria de la disponible en el ordenador (ya que hay que tener en cuenta que gran parte de esta será destinada a la ejecución del juego). Por otro lado, se ha desestimado la posibilidad de utilizar una estructura de acceso a fichero en el que se realicen almacenamientos de datos no utilizados recientemente, puesto que su implementación resultaría muy costosa y no entra dentro del ámbito de estudio de este proyecto.

- Problemas para cuantificar estados. Dado que los estados del modelo estarán compuestos por diversos valores procedentes de sensores del bot (los cuales en muchos casos son medidos mediante intervalos de números reales), el número de estados posibles es insostenible.

- Problemas para ceñirse al algoritmo en los primeros ciclos de ejecución. Este problema deriva de la discriminación de aquellas acciones previamente no utilizadas y que se pueden realizar desde un estado concreto. Siempre que alguna de las acciones que sí han sido visitadas, haya recibido un refuerzo positivo, técnicamente tendrán una prioridad mayor a la hora de realizarse que las que aún no se han probado (con lo que se puede incurrir en una toma de decisiones no óptima). En el apartado sobre *Implementación de Prototipo* se plantearán mecanismos para evitar este problema.

Sin embargo, esta alternativa también presenta ciertas ventajas frente a la técnica empleada para desarrollar el prototipo anterior:

- Nivel cognitivo superior. El nivel de la escala ConsScale alcanzado mediante esta solución es el adaptativo, mientras que en el caso anterior, sólo se consiguió alcanzar una respuesta de tipo reactivo.

- Uso de memoria. El hecho de variar el comportamiento del bot de manera dinámica obliga a almacenar los valores tras la realización de una partida, de forma que la evolución pueda ser mantenida en sucesivas ejecuciones. Para ello, se debe implementar una memoria donde se almacene dicha información tras una partida, y de la que se restaure al comienzo de la siguiente.

En cuanto al riesgo que presenta esta alternativa enfocada al modelo del estudio, únicamente existe un problema: la incapacidad para aplicar un refuerzo asociado a una acción, con la plena seguridad de que es la correcta. Esto se debe a que el paso de una acción a otra se realiza de manera síncrona, cuando se llama a una función determinada, mientras que la notificación de una muerte (propia o ajena) es un evento asíncrono que el agente recibe en forma de mensaje en momentos indefinidos de la partida. Suele darse con frecuencia, que en el momento que se recibe dicha notificación, ya se ha transitado de la acción que debería sufrir el refuerzo a la siguiente, con lo que la única forma de aplicar el

refuerzo de una manera totalmente segura sería llevando un control de tiempo que vincule el instante de tiempo en que se lanzó una determinada notificación, con la acción que se estaba realizando en ese momento. El problema es que Gamebots no aporta ningún mecanismo que permita obtener ambos datos, por lo que el riesgo reside en que el refuerzo se asigne a una tupla estado-acción incorrecta. Aún así, ésta resulta una alternativa interesante para generar un prototipo ya que, tras una serie de partidas que permitan al bot evolucionar convenientemente, el uso de esta técnica puede resultar interesante para igualar e incluso superar los resultados obtenidos mediante la anterior solución.

Implementación de Prototipo

A lo largo de este apartado se detallarán todos los aspectos relativos al desarrollo de un prototipo que aplique el algoritmo de Q Learning.

Estados definidos

Para poder trabajar con el algoritmo de Q Learning, es preciso llevar a cabo la formalización de un conjunto de estados que serán evaluados durante la ejecución. En este caso, cada uno de los estados estará formado por la combinación de los distintos valores posibles de los sensores más significativos del agente autónomo. Dichos sensores son:

Nombre del Sensor	Tipo de Dato	Descripción
Disparo	Binario	Especifica si el agente está disparando actualmente o no.
Daño	Binario	Denota si el agente está recibiendo disparos de algún otro bot o no.
Distancia al Enemigo	Real	Longitud que separa en el espacio al agente autónomo de cualquier enemigo.
Salud	Real	Valor que expresa el nivel de vida restante del agente.
Armadura	Real	Nivel de protección con que cuenta el bot.
Armas	Entero	Número de armas distintas que porta el agente.
Munición Primaria	Entero	Nivel de munición de tipo primaria del arma que actualmente porta el agente.
Munición Secundaria	Entero	Nivel de munición alternativa del arma que utiliza actualmente el bot.

Tabla 4 Tipos de datos de los sensores del agente

El correcto procesamiento de estos valores dota al agente autónomo de un gran control sobre la situación del entorno (y su propio organismo) en cada instante de la partida. Sin embargo, existen problemas derivados de este planteamiento para definir los estados de la tabla, ya que muchos de los valores arriba citados son atributos continuos y de tipo real, con grandes intervalos de acción. El problema pues, deriva de la necesidad de generar todas las combinaciones posibles de estos datos entre sí, ya que el uso de una única variable real supone decenas de miles de combinaciones. Por todo ello, es necesario realizar una discretización de los campos implicados, de tal manera que la combinación total de valores conforme un conjunto de estados asumible para formar una tabla que debe permanecer en memoria durante todo el tiempo de ejecución del bot. Dicha simplificación se realiza de la siguiente manera:

Nombre del Sensor	Tipo de Dato	Valores Posibles
Disparo	Binario	• 0, si el agente no está disparando • 1, en caso contrario
Daño	Binario	• 0, si el agente no está siendo disparado • 1, en caso contrario
Distancia al Enemigo	Ternario	• 0, si no hay ningún enemigo en el campo de visión del agente • 1, si el enemigo se encuentra cerca del agente • 2, si el agente se encuentra lejos
Salud	Ternario	• 0, si el nivel de salud del agente es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto
Armadura	Ternario	• 0, si el nivel de armadura del agente es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto
Armas	Binario	• 0, si el agente no tiene suficientes armas en su inventario • 1, en caso contrario
Munición Primaria	Ternario	• 0, si el nivel de munición primaria del arma actual es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto
Munición Secundaria	Ternario	• 0, si el nivel de munición secundaria del arma actual es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto

Tabla 5 Discretización de los valores que componen el estado del bot

De esta forma, las posibles combinaciones de estados se reducen de manera ostensible, resultando únicamente 1944 posibles estados.

Estructura de tablas empleada

Con el fin de aplicar el algoritmo de Q Learning sobre este segundo prototipo, se ha decidido utilizar una única tabla de refuerzos que englobe todos los posibles estados y sus correspondientes acciones. De este modo, sólo podrá realizarse una acción de forma simultánea (aspecto que simplifica bastante la asignación de los refuerzos).

Gestión de Memoria

A la hora de gestionar el conocimiento adquirido por el bot a lo largo de las diferentes partidas jugadas, se utiliza un archivo de texto plano en el que se almacenan los valores de la tabla Q, así como los valores de una tabla de iguales dimensiones, encargada de determinar qué posiciones de la tabla Q no han sido exploradas aún. El formato del archivo es el siguiente:

Q Values:

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$q_{1,1}$	$q_{1,2}$	…	$q_{1,n}$
<estado 2>	$q_{2,1}$	$q_{2,2}$	…	$q_{2,n}$
…	…	…	…	…
<estado m>	$q_{m,1}$	$q_{m,2}$	…	$q_{m,n}$

Q-Visited (boolean):

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$v_{1,1}$	$v_{1,2}$	…	$v_{1,n}$
<estado 2>	$v_{2,1}$	$v_{2,2}$	…	$v_{2,n}$
…	…	…	…	…
<estado m>	$v_{m,1}$	$v_{m,2}$	…	$v_{m,n}$

Donde *q* representa el valor correspondiente acumulado de cada par de elementos estado-acción y *v* es un valor binario que denota si dicho par ha sido visitado durante las partidas anteriores (0, en caso negativo; 1, en caso afirmativo).

De este modo, es posible almacenar los valores de las tablas al finalizar la ejecución del bot, de una forma inteligible y fácilmente restaurable al iniciar una nueva partida.

Aplicación del Algoritmo en Tiempo de Ejecución

Durante la ejecución de una partida, el agente realiza llamadas constantes al algoritmo de Q-Learning, de manera que en cada ciclo de ejecución, se llevan a cabo labores de actualización de estados, selección de acciones a realizar y aplicación de refuerzos (si procede), por medio de la fórmula definida en el apartado de *Aprendizaje por refuerzo* de la sección de *Estado del Arte*.

Para actualizar el estado, se definen en el código unos métodos encargados de evaluar los niveles de los sensores involucrados y, en cada caso concreto, se discretizan los valores. Los criterios utilizados para cada uno de los atributos discretizados son los siguientes:

Sensor	Valores Posibles	Criterios
Disparo	• 0, si el agente no está disparando • 1, en caso contrario	Consulta del método isShooting() del API de Gamebots
Daño	• 0, si el agente no está siendo disparado • 1, en caso contrario	Consulta del método isBeingShot() del API de Gamebots
Distancia al Enemigo	• 0, si no hay ningún enemigo en el campo de visión del agente • 1, si el enemigo está cerca del agente • 2, si el agente se encuentra lejos	Si el método distanceInSpace (posicion1, posicion2) del API de Gamebots devuelve un valor menor de 500, se considera una distancia corta
Salud	• 0, si el nivel de salud del agente es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto	Si el método getAgentHealth() del API de Gamebots devuelve un valor menor de 35, el nivel es bajo; si está entre 35 y 70, nivel medio; y por encima de 70, es considerado nivel alto
Armadura	• 0, si el nivel de armadura del agente es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto	Si el método getAgentArmor() del API de Gamebots devuelve un valor menor de 35, el nivel es bajo; si está entre 35 y 70, nivel medio; y por encima de 70, es considerado nivel alto
Armas	• 0, si el agente no tiene suficientes armas en su inventario • 1, en caso contrario	Si el número de armas en el inventario es menor que la mitad de la capacidad del mismo, el nivel de armas es bajo; sería alto, en caso contrario
Munición Primaria	• 0, si el nivel de munición primaria del arma actual es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto	El nivel será bajo si hay menos de un tercio de la capacidad máxima; medio, si está entre uno y dos tercios; y alto en el resto de casos
Munición Secundaria	• 0, si el nivel de munición secundaria del arma actual es bajo • 1, si el nivel es intermedio • 2, si el nivel es alto	El nivel será bajo si hay menos de un tercio de la capacidad máxima; medio, si está entre uno y dos tercios; y alto en el resto de casos

Tabla 6 Criterios utilizados para el uso de valores discretizados

A la hora de elegir una nueva acción, se comprobará el estado actual mediante la discretización de los parámetros anteriormente analizados y, de entre todas las acciones disponibles, se cogerá la primera de las no visitadas aún desde ese estado. En caso de que todas lo hayan sido en algún momento, se cogerá la de mayor valor q y, en caso de que existieran varios máximos, se realizará una elección aleatoria entre todas ellas.

Finalmente, la asignación de refuerzos se realiza a través de cuatro tipos de notificación de evento distintos. Estos son:

- Jugador herido. Este mensaje implica que un jugador de la partida, distinto del propio agente autónomo, ha sido alcanzado por una bala. Al tratarse de un proyecto pensado para realizar partidas de uno contra uno, se asume que el jugador que ha ejecutado el disparo es el propio agente, por lo que será premiado con la bonificación pertinente sobre el valor q correspondiente al par de estado-acción actuales, calculado con la fórmula de Q Learning.

- Jugador muerto. Al igual que en el caso anterior, se asume que el ejecutor de la muerte es el propio agente, por lo que será premiado con la bonificación resultante de aplicar la fórmula de Q Learning.

- Agente herido. Al recibir un disparo, el agente recibe una penalización (calculada en la fórmula de Q Learning) sobre el valor previo de q para el par estado-acción actual.

- Agente muerto. La muerte del propio agente es la situación menos deseada del juego. Por ello, está se castigará restando la puntuación que especifique la fórmula de Q Learning al valor q.

Es importante hacer hincapié en la problemática que conlleva este tipo de refuerzos, puesto que existe el riesgo de asignar refuerzos a pares estado-acción incorrectos. Además, la aplicación de este prototipo en partidas con más de un contrincante conllevará la aplicación de refuerzos positivos en situaciones equivocadas con total seguridad.

Estructura de Clases

A continuación se muestra el diagrama de clases correspondiente a la implementación de este segundo prototipo:

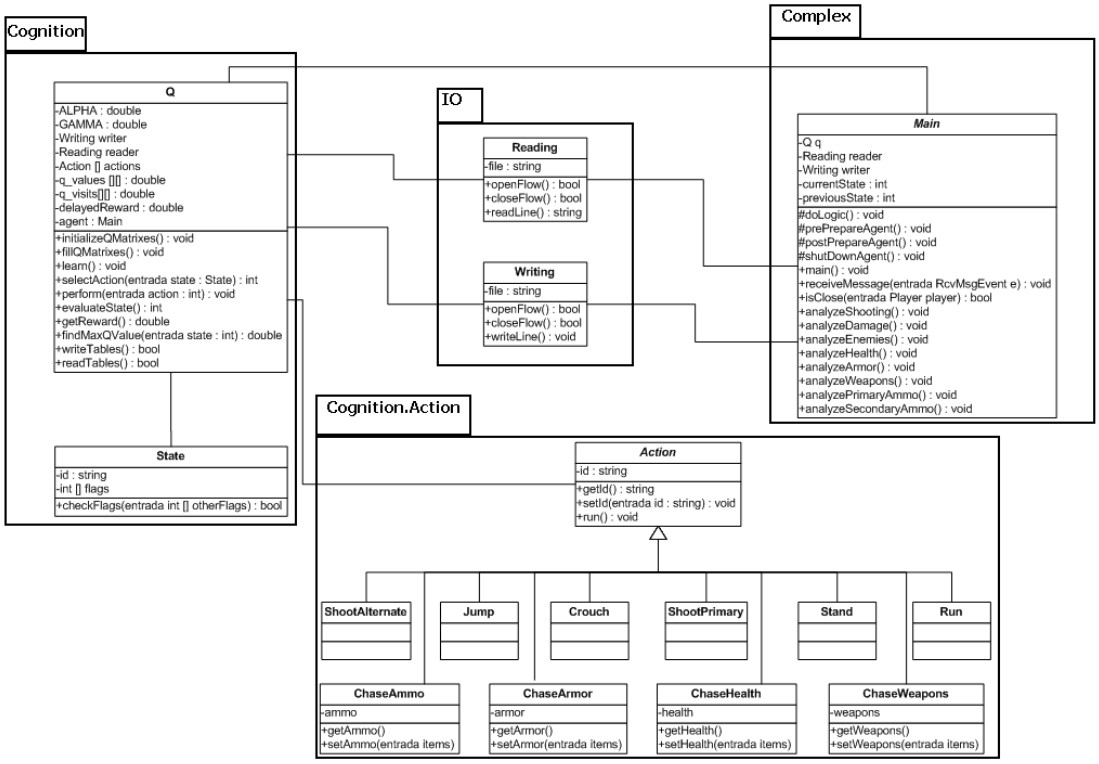

Figura 14 Diagrama de clases del agente basado en Q Learning

Como se puede observar en la *Figura 14*, existen cuatro paquetes bien diferenciados dentro del código del proyecto Pogamut:

- Paquete Principal. En el se encuentra la clase *Main* que será ejecutada al utilizar al bot en una partida. Además de los métodos arriba comentados para obtener valores discretos de los sensores utilizados para generar los diversos estados, existen cuatro métodos importantes:
    - doLogic(). Método necesario para la ejecución de la clase, que es llamado constantemente durante el periodo en que el bot participa en la partida. Contiene toda la funcionalidad que el agente debe reproducir a lo largo del juego.
    - prePrepareAgent(). Método necesario para la ejecución de la clase, que contiene inicializaciones de atributos de la clase. Es llamado una única vez antes de comenzar a jugar.
    - postPrepareAgent(). Método necesario para la ejecución de la clase, que contiene inicializaciones de parámetros relativos a elementos de comunicación del agente. Se llama una vez antes de comenzar la partida y, en este caso, incluye las definiciones

de los eventos que deben ser notificados, y la inicialización de la matriz Q.

- shutDownAgent(). Se encarga de realizar actividades al finalizar la ejecución del agente. En este caso, se realiza el almacenamiento de información en el fichero que contiene la memoria del agente.

- Paquete IO. Contiene las clases encargadas de leer y escribir en fichero los datos correspondientes a la memoria del agente autónomo. Para ello, se facilita funcionalidad como escribir línea, leer línea, abrir y cerrar flujo de datos de un fichero, etc.

- Paquete Cognition. Este paquete contiene toda la información relativa a la aplicación del algoritmo Q Learning. Se divide en los siguiente elementos:

  - Clase Q. Contiene la funcionalidad necesaria para inicializar la matriz Q (partiendo de cero, o restaurándola desde un fichero), aplicar el algoritmo convenientemente y manipular determinados elementos de la matriz Q.

  - Clase State. Contiene la estructura de estado formada por los diversos sensores del robot. Permite manipular valores de sensores concretos y realizar comparaciones con otros estados.

  - Paquete Action. Contiene la clase abstracta *Action* que especifica que todas sus clases hijas deben sobrescribir convenientemente su método *run()*. De este modo, cada una de las hijas representa una de las acciones a realizar por el agente.

Evaluación del Prototipo

Al igual que con el prototipo anterior, en esta sección se realizará una serie de combates sucesivos entre dos agentes semejantes, que utilicen la misma técnica para desarrollar su comportamiento (en este caso, Q Learning). A diferencia del caso anterior, ahora pueden darse diferencias entre los competidores, ya que se pueden variar los valores de las constantes α y γ del algoritmo de Q Learning para observar cómo repercute en el comportamiento de los bots y poder sacar conclusiones al respecto.

Respecto a las pruebas de agentes con Q Learning combatiendo contra individuos procedentes de otras implementaciones, se hablará en detalle en la sección *Diseño de Experimentos y Resultados*.

Comenzaremos las pruebas obteniendo conclusiones acerca de la repercusión que tiene α en el comportamiento de los agentes. Por ello, dejaremos un valor fijo de γ para todos los rivales. Los resultados de los distintos encuentros son los siguientes:

Prueba 1	
Tiempo Transcurrido:	2 horas
Valor γ:	0.8
Valor α Agente Ganador:	0.5
Valor α Agente Perdedor:	1
Marcador Agente Ganador:	8
Marcador Agente Perdedor:	3
Tasa de Igualdad:	2.33

Tabla 7 Prueba 1 cambiando el valor de alpha

El motivo de haber agotado el tiempo de juego es que en un momento de la ejecución, ambos jugadores concluyeron que la actitud más recomendable era quedarse erguidos. Al haber exclusividad de movimiento (es decir, no se pueden realizar varias acciones simultáneamente), el juego quedó paralizado.

Respecto a los datos obtenidos, no son concluyentes aunque dejan vislumbrar una ligera mejora cuando el valor de α es bajo.

Prueba 2	
Tiempo Transcurrido:	2 horas
Valor γ:	0.8
Valor α Agente Ganador:	0.1
Valor α Agente Perdedor:	0.5
Marcador Agente A:	4
Marcador Agente B:	0
Tasa de Igualdad:	3

Tabla 8 Prueba 2 cambiando el valor de alpha

Al igual que en la prueba anterior, en esta segunda prueba ambos jugadores quedan paralizados al concluir que era recomendable quedarse quieto. En este caso vuelve a darse un resultado contundente del agente con menor valor de alpha, por lo que se puede deducir que a rangos bajos de esta constante, el rendimiento del bot mejora.

Respecto a la inmovilidad de los agentes, es un fenómeno comprensible teniendo en cuenta que, para evitar que uno de ellos pueda partir en desventaja con respecto al otro, las ejecuciones se han realizado sin tener en cuenta un proceso de aprendizaje previo. Es decir, ninguno de los dos tiene experiencias previas vividas al empezar el combate. En las pruebas contempladas en la

sección *Diseño de Experimentos y Resultados* se tendrá en cuenta el uso de la memoria creada en combates previos para observar el comportamiento del agente una vez entrenado.

A continuación, se fijará el valor de α a 1, y se variará el valor de γ para observar las repercusiones que tiene sobre la capacidad cognitiva de los agentes.

Prueba 3	
Tiempo Transcurrido:	2 horas
Valor α:	1
Valor γ Agente Ganador:	0.5
Valor γ Agente Perdedor:	0.8
Marcador Agente Ganador:	5
Marcador Agente Perdedor:	4
Tasa de Igualdad:	1

Tabla 9 Prueba 1 variando el valor de gamma

En este caso, se ha vuelto a producir el fenómeno por el cual, ambos agentes quedan paralizados ante la alternativa de mantenerse erguidos, lo que nos lleva a pensar que es necesario aplicar una mejora al modelo para evitarlo en posteriores ejecuciones.

La tasa de igualdad es 1, por lo que los resultados, tras dos horas de partida, no resultan concluyentes a la hora de evaluar cuál de los valores de γ es más efectivo.

Prueba 4	
Tiempo Transcurrido:	2 horas
Valor α:	1
Valor γ Agente Ganador:	0.5
Valor γ Agente Perdedor:	0.1
Marcador Agente Ganador:	5
Marcador Agente Perdedor:	1
Tasa de Igualdad:	1

Tabla 10 Prueba 2 variando el valor de gamma

Este caso se contradice la hipótesis que podría generarse de la prueba anterior, por la que agentes con un valor de γ menor podrían resultar más eficientes, ya que en este caso, es el agente de mayor valor de γ el que consigue

ganar la partida. Además, el resultado es más rotundo que en el caso anterior, lo que nos lleva a pensar que no se debe únicamente a un cúmulo de casualidades o a aspectos meramente estadísticos.

Los agentes siguen bloqueándose tras un cierto tiempo de ejecución, por lo que se hace imprescindible aplicar una mejora al funcionamiento de este prototipo, o plantearse la posibilidad de aplicar otro tipo de técnica que resulte más efectiva de cara a obtener buenos resultados en combate.

Ciclo 3: Evolución del Prototipo del Ciclo 2

El modelo anterior, aunque interesante en términos de prestaciones cognitivas, tenía grandes carencias, en su mayoría procedentes del hecho de no poder realizar de manera simultánea actividades compatibles entre sí (por ejemplo, no se puede disparar mientras se recorre el escenario saltando).

En este tercer ciclo se plantea la posibilidad de mejorar el rendimiento del agente con Q Learning, haciendo posible que se realicen acciones compatibles de manera simultanea, sin dejar de lado la técnica aplicada en el ciclo anterior. Para llevar a cabo este objetivo, se mantendrá la misma funcionalidad analizada en el *Ciclo 2: Desarrollo mediante Aprendizaje por Refuerzo*.

Análisis de Requisitos

En este caso, la especificación de requisitos no varía respecto al prototipo anterior (ver sección sobre *Análisis de Requisitos* del *Ciclo 2: Desarrollo mediante Aprendizaje por Refuerzo*).

Riesgos, Dificultades y Oportunidades de la Arquitectura

El objetivo de este tercer desarrollo es, como ya se ha comentado, realizar una mejora del modelo anterior que permite poder realizar varias acciones simultaneas, sin renunciar por ello a usar el algoritmo Q Learning.

La alternativa propuesta consiste en desarrollar tres tablas Q en lugar de una sola. Cada una de estas tablas contendrá todas las posibles combinaciones de estados pero, tan sólo un subconjunto del total de acciones. Las acciones asignadas a cada tabla estarán agrupadas por tipo de funcionalidad. Es decir, se tendrá una tabla que posea como acciones a realizar, sólo aquellas relacionadas con el ataque (uso de disparo primario, secundario y dejar de disparar); otra, sólo tendrá acciones relativas al movimiento horizontal (recorrer el escenario y buscar ítems de un tipo determinado); finalmente, la tercera tabla contendrá las acciones relativas al movimiento vertical del agente (saltar, agacharse y erguirse).

Los inconvenientes de aplicar esta mejora a la técnica de Q Learning son los mismos que en el ciclo anterior, puesto que siguen teniéndose dificultades para dar cabida a la gran demanda de memoria que requiere el uso de tablas de tamaño grande y fijo; además, sigue existiendo el problema de un aprendizaje defectuoso en etapas tempranas del juego, en las que la suerte juega un papel fundamental a la hora de que una jugada teóricamente acertada, sea valorada como tal.

Sin embargo, esta mejora incorpora nuevas ventajas:

- Introduce la posibilidad de realizar varias acciones complementarias de forma simultánea.
- Los requisitos de memoria apenas varían, ya que el número de elementos *q* que conforman las tres tablas de este nuevo modelo, será

el mismo que el utilizado para el modelo anterior, ya que el número de estados y acciones sigue siendo el mismo aunque sea gestionado de forma diferente.

Por todo ello, se considera que esta mejora del prototipo implementado en la segunda fase del ciclo de vida, puede resultar interesante de cara a dotar al nuevo individuo de un mayor grado de inteligencia.

Implementación de Prototipo

El diseño llevado a cabo para desarrollar el modelo de agente con Q Learning ha resultado altamente adaptable a cambios como el presente, en el que se pretende aplicar nuevas mejoras.

En cuanto a la estructura de clases, se ha construido de la siguiente manera:

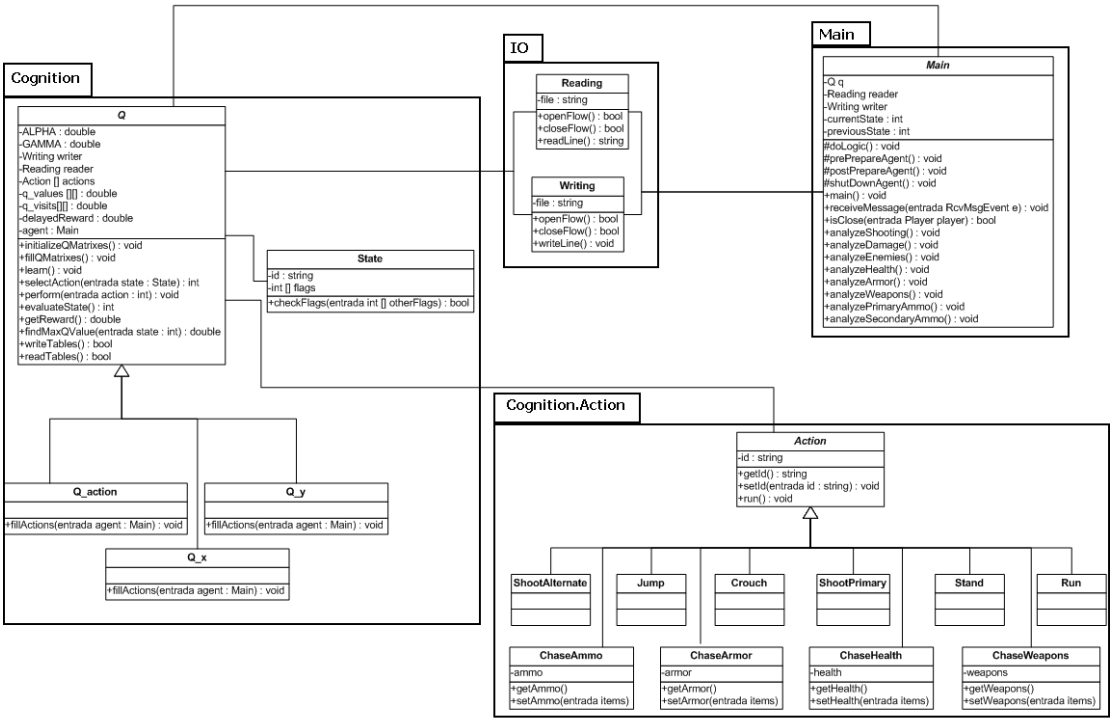

Figura 15 Diagrama de clases del agente mejorado basado en Q Learning

La única diferencia sustancial con el diseño planteado para el segundo prototipo de este estudio (ver *Estructura de Clases* del prototipo 2), es que ahora existe una jerarquía de clases Q en la cual, toda la funcionalidad genérica está implementada en la clase abstracta *Q*, mientras que cada una de las clases hijas se encargas de definir de forma específica cuáles serán las acciones a llevar a cabo durante la ejecución de su algoritmo. De este modo, la mayor parte de la funcionalidad sigue estando centralizada, mientras que se adquiere un diseño adaptable a posibles modificaciones que se quisieran hacer en un futuro (como añadir nuevos módulos de acciones, eliminar alguno de los ya implementados, etc). Como ya se ha descrito en los casos de uso de esta alternativa, una consecuencia de esta jerarquía, es la inclusión de nuevas acciones que permiten:

- Parar de disparar. En un módulo de ataque que ofrece dos funciones de disparo, debe haber otra acción que permita detener el fuego en un momento concreto para ahorrar munición.

- Perseguir a un enemigo. Supone una extensión de las acciones en las que se recorre el escenario buscando un ítem determinado y puede resultar útil en situaciones en las que se debe atacar al adversario con un arma de contacto (*Melee Weapon*).

- Parar de correr. Implica detener el movimiento horizontal del agente. Con ello se pretende mejorar el resultado de situaciones en las que mientras se apunta y dispara a un adversario, el cuerpo del agente continúa recorriendo un circuito que, en ocasiones, puede alejarle de su objetivo o rotar el arma sacando al enemigo del campo de visión.

Aún así, la modificación del código ha sido mínima, variando únicamente los siguientes aspectos:

- El bot trabaja con tres tablas Q en lugar de una. Por tanto es preciso llevar a cabo ejecuciones de los algoritmos Q Learning que modifiquen cada una de ellas de manera secuencial y en cada ciclo de ejecución. De este modo los tres comportamientos funcionales variarán constantemente, pudiendo coexistir en el tiempo.

- Inclusión de nuevas clases hijas dependientes de la clase abstracta *Action*. De este modo, se da cabida a las nuevas acciones a realizar por el agente.

- La gestión de memoria en fichero ha tenido que ser modificada para hacer frente a los cambios de la estructura de clases. Al no existir una escritura/lectura centralizadas del fichero de memoria, cada tabla se encarga de almacenar sus datos en el fichero de manera secuencial. Por ello, el formato del fichero utilizado en esta mejora es el siguiente:

```
#### Action section ####
```
Q Values:

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$q_{1,1}$	$q_{1,2}$	…	$q_{1,n}$
<estado 2>	$q_{2,1}$	$q_{2,2}$	…	$q_{2,n}$
…	…	…	…	…
<estado m>	$q_{m,1}$	$q_{m,2}$	…	$q_{m,n}$

Q-Visited (bolean):

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$v_{1,1}$	$v_{1,2}$	…	$v_{1,n}$
<estado 2>	$v_{2,1}$	$v_{2,2}$	…	$v_{2,n}$
…	…	…	…	…
<estado m>	$v_{m,1}$	$v_{m,2}$	…	$v_{m,n}$

Horizontal Movement section

Q Values:

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$q_{1,1}$	$q_{1,2}$	…	$q_{1,n}$
<estado 2>	$q_{2,1}$	$q_{2,2}$	…	$q_{2,n}$
…	…	…	…	…
<estado m>	$q_{m,1}$	$q_{m,2}$	…	$q_{m,n}$

Q-Visited (bolean):

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$v_{1,1}$	$v_{1,2}$	…	$v_{1,n}$
<estado 2>	$v_{2,1}$	$v_{2,2}$	…	$v_{2,n}$
…	…	…	…	…
<estado m>	$v_{m,1}$	$v_{m,2}$	…	$v_{m,n}$

Vertical Movement section

Q Values:

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$q_{1,1}$	$q_{1,2}$	…	$q_{1,n}$
<estado 2>	$q_{2,1}$	$q_{2,2}$	…	$q_{2,n}$
…	…	…	…	…
<estado m>	$q_{m,1}$	$q_{m,2}$	…	$q_{m,n}$

Q-Visited (bolean):

States\Actions	*<acción 1>*	*<acción 2>*	*…*	*<acción n>*
<estado 1>	$v_{1,1}$	$v_{1,2}$	…	$v_{1,n}$
<estado 2>	$v_{2,1}$	$v_{2,2}$	…	$v_{2,n}$
…	…	…	…	…
<estado m>	$v_{m,1}$	$v_{m,2}$	…	$v_{m,n}$

Evaluación del Prototipo

Para llevar a cabo una primera evaluación del rendimiento alcanzado por este agente será preciso compararlo con su predecesor. Por ello, se realizará un combate entre el agente de Q Learning y el agente de Q Learning mejorado, introduciendo en ambos los mismos valores para α y γ. Es preciso comentar que esta prueba es realizada sin entrenamiento previo de ninguno de los agentes, por lo que únicamente se podrán arrojar conclusiones respecto a la capacidad de cada uno de ellos para adaptarse a un medio sin experiencia previa. Para llevar a cabo un veredicto pleno acerca del rendimiento de este agente, será preciso esperar a realizar las pruebas de la sección de *Diseño de Experimentos y Resultados*, para la cual se proporcionará un entrenamiento de 10 partidas a

todos aquellos prototipos adaptativos que se utilicen. Una vez aclarado este aspecto, los resultados obtenidos son los siguientes:

Tiempo Transcurrido:	2 horas
Marcador Q Learning Básico:	5
Marcador Q Learning Mejorado:	2
Tasa de Igualdad:	2
α Utilizada:	0.1
γ Utilizada:	0.5

Tabla 11 Resultados de la prueba de mejora del bot Q Learning

Como se puede observar en la *Tabla 11*, el bot implementado con el algoritmo Q Learning mejorado ha sido superado por su predecesor. No obstante, si se analiza la situación en detalle, no resulta un resultado tan inverosímil. Estos son los principales motivos de la derrota:

- Pese a que el nuevo agente consta de una mayor funcionalidad, ésta también implica un mayor abanico de posibilidades que deberá probar de cara a analizar que situaciones son las más deseables. Por tanto, el hecho de contar con más funcionalidad sólo permite que exista la posibilidad de modelar correctamente el comportamiento deseado, pero no lo garantiza.

- El hecho de realizar tres actividades de forma simultanea, no sólo triplica las oportunidades de éxito, sino también las de fracaso de un comportamiento.

- El problema de parálisis estudiado en el prototipo de la segunda fase, persiste en esta tercera etapa, por lo que una vez que ambos contrincantes entran en una fase de estancamiento, la partida agotará el tiempo máximo sin que ninguno de ellos modifique el marcador obtenido hasta ese momento. Este factor se debe obviamente a que el agente no ha sido previamente entrenado.

Una vez observados todos los posibles motivos de que esta nueva versión del agente no haya mejorado los resultados previos, se concluye que es necesario llevar a cabo un desarrollo nuevo que utilice una técnica de IA totalmente distinta a la actual o, que al menos, la modifique de manera ostensible.

Ciclo 4: Desarrollo mediante Sistema Híbrido

En esta nueva etapa del ciclo de vida del software, se pretende llevar a cabo el desarrollo de un nuevo prototipo que revolucione el comportamiento observado hasta ahora en ciclos anteriores del modelo.

El objetivo consiste en extraer todas las ventajas de los modelos anteriormente implementados y combinarlos en un prototipo híbrido, que incluya un sistema experto y un algoritmo de Q Learning.

Inicialmente, se reducirá la funcionalidad disponible respecto al anterior prototipo, ya que resultó no ser de utilidad. "Detectar invulnerabilidad" y "Correr hacia el enemigo" serán los únicos casos de uso eliminados de este modelo respecto a los prototipos basados en aprendizaje por refuerzo (ver *Ciclo 2: Desarrollo mediante Aprendizaje por Refuerzo*).

Análisis de Requisitos

Al igual que en la etapa anterior, será preciso llevar a cabo una modificación de los requisitos de software definidos para el prototipo inicial (ver *Ciclo 1: Desarrollo mediante Sistemas Expertos*). Los requisitos sujetos a cambios serán los identificados como *RF* 7, *RF* 8 y *RNF* 14 y quedarán de la siguiente manera:

Identificador	RF007				
Nombre	Nivel de Cognición				
Actores	Agente				
Descripción	El agente debe alcanzar una capacidad cognitiva que satisfaga al menos, el nivel adaptativo de la escala ConsScale				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 11 Modificación del RF007 del ciclo 1

Identificador	RF008				
Nombre	Técnica de IA Aplicada				
Actores	Agente				
Descripción	El sistema aplicará las técnicas de sistemas expertos y aprendizaje por refuerzo para dotar al agente de cierto grado de inteligencia				
Origen	RU002				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RF 12 Modificación del RF008 del ciclo 1

Identificador	RNF014				
Nombre	Especificaciones sobre Técnicas Aplicadas				
Actores	Sistema				
Descripción	El algoritmo de aprendizaje por refuerzo a utilizar será Q-Learning y el sistema experto será aplicado mediante el uso de sentencias condicionales				
Origen	RU002, RF008				
Verificable	Sí		**Claro**	Sí	
Prioridad	5	**Necesidad**	Esencial	**Estabilidad**	5

RNF 16 Modificación del RNF014 del ciclo 1

Riesgos, Dificultades y Oportunidades de la Arquitectura

A la hora de llevar a cabo este desarrollo híbrido, existe el riesgo de heredar los inconvenientes observados en los modelos de los que deriva. Por ello, será necesario limitar el uso de cada una de las técnicas con el fin de evitar dichas situaciones indeseadas.

La principal dificultad que se plantea es, por tanto, hallar el equilibrio en el uso de cada uno de los mecanismos planteados. A continuación, se resumen los pros y contras de cada uno de ellos, para poder comprender mejor el problema de integración que hay que resolver:

	Sistema Experto	**Aprendizaje por Refuerzo**
Nivel Cognitivo	Reactivo	Adaptativo
Uso de memoria	No	Sí
Memoria requerida	Nula	Mucha
Problemas durante adaptación inicial	No	Sí

Tabla 12 Características de las técnicas de IA empleadas

Como puede observarse, el uso del algoritmo Q Learning aporta la posibilidad teórica de alcanzar un nivel cognitivo superior. No obstante, plantea ciertos problemas iniciales de adaptación al medio, dado que todos los refuerzos son muy similares y la elección adecuada de una acción dependerá, en caso de entrenar contra un agente de complejidad semejante, de la suerte del agente durante los primeros compases de la partida; y en caso de jugar contra un entrenador humano, de la pericia del mismo para conseguir que el agente logre adaptarse poco a poco a las condiciones del juego. Además, el algoritmo de Q Learning realiza un gran consumo innecesario de recursos al tener que generar una matriz con todos los pares estado-acción posibles.

Por ello, la solución implementada en esta etapa del ciclo de vida, debería cumplir con un modelo adaptativo que proporcione una memoria en la que poder almacenar la información cognitiva de una partida a otra y, que sea capaz de evitar los problemas iniciales propios de un modelo que evoluciona partiendo de cero. De esta manera, se conseguiría alcanzar un modelo cognitivo capaz de contar con un grado de complejidad inicial que le permita maniobrar con la suficiente coherencia desde el primer ciclo de ejecución de la partida.

Implementación de Prototipo

Con el fin de evitar problemas con el consumo excesivo de memoria, se sustituirá el algoritmo de Q Learning por otro cuya funcionalidad sea similar, pero permita definir las situaciones a medida que se van desarrollando. De este modo se consigue descongestionar la memoria del ordenador en tiempo de ejecución. Para ello se plantea la siguiente estructura de clases:

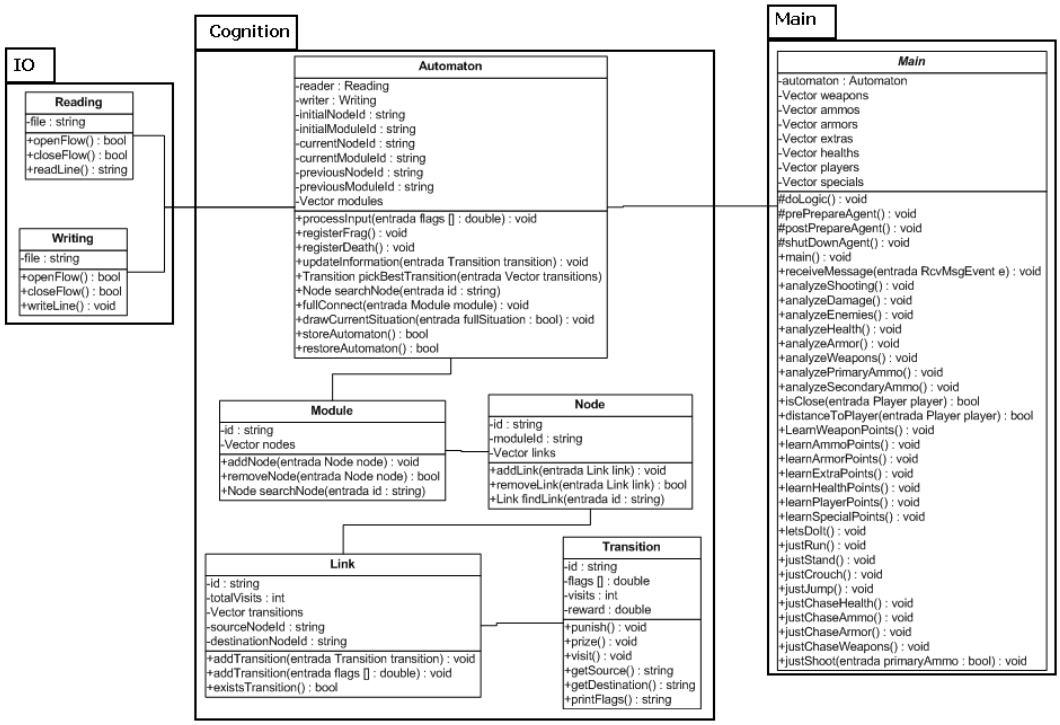

Figura 16 Diagrama de clases de agente híbrido básico

Como se puede observar, se ha sustituido las diversas implementaciones de la clase Q, por una máquina de estados finita. Dicha máquina cumple con los siguientes aspectos:

- Existe una única máquina de estados que modela la toma de decisiones del agente.
- Una máquina de estados está formada por un conjunto de módulos. Dichos módulos albergan funcionalidad relacionada entre sí por su propósito (es decir, habrá un módulo de ataque que abarque acciones como disparar con función primaria y secundaria; otro módulo encargado de realizar acciones neutrales, como correr, etc).
- Un módulo está formado por un conjunto de nodos. Cada uno de estos nodos representa una acción concreta del módulo.
- El agente puede pasar de realizar una acción a otra, por medio de los enlaces que unen a todos los nodos de la máquina de estados, con todos los demás (y consigo mismo). Dichos enlaces tienen carácter unidireccional, por lo que para todo nodo A existirá un enlace con todo nodo B y para todo nodo B existirá un enlace con todo nodo A.
- Cada vez que se utiliza un enlace para realizar una acción diferente, se genera una transición ligada al enlace concreto. Dicha transición contendrá los niveles concretos de los sensores del agente que se poseían en el momento de la creación de dicha transición, así como el número de veces que se ha dado dicha transición y el valor de refuerzo acumulado por la misma.

- Cuando sea preciso transitar de un nodo a otro de la máquina, y no haya registrada ninguna transición con los niveles de sensores concretos que se tienen en la actualidad, no sólo se generará la transición en un enlace concreto que parta del nodo actual, sino en todos aquellos que partan del éste hacia otros nodos. De este modo, se facilita el proceso de evaluación del refuerzo de todos los enlaces en sucesivas consultas para cambiar de acción.

Por último, es preciso destacar que los refuerzos aplicados en este sistema, se llevan a cabo sobre las transiciones que conducen al comportamiento de un nodo al siguiente. De este modo, si mientras se está corriendo, se detecta un enemigo, y se transita al nodo disparo y el agente consigue eliminarlo, se premiará a la transición "*correr → disparar*". Este aspecto implica hacer partícipe del premio, no sólo al nodo nuevo, sino también al anterior.

Mecanismo de sincronización

En el modelo anterior, que utilizaba la técnica de Q Learning, existía un inconveniente a la hora de asignar los refuerzos dentro de la matriz Q, ya que la notificación de muertes durante la partida, era gestionada por un método de carácter asíncrono, que notificaba distintos mensajes con un identificador del tipo al que pertenecía. En dichas estructuras de mensajes, no había ningún campo que contuviese el instante de tiempo en que se produjo el evento y, por tanto, no era posible saber qué estado y acción se estaban ejecutando en el momento en el que se generó dicha notificación, dado que no tenían que ser necesariamente las mismas que se ejecutaban al ser recibida.

Por eso mismo, se ha aportado una mejora de este control de sincronización de forma que, aunque no sea posible desarrollar la asignación de refuerzos con la máxima precisión, al menos se aplica dentro de un conjunto de situaciones más acotado. El mecanismo de sincronización implementado consiste en recordar en cada momento, cuál fue el último nodo visitado del módulo de ataque. De esta manera, al recibirse una notificación de muerte ajena, se asignará el refuerzo positivo a la transición que condujo a dicho nodo. Sigue sin ser una técnica completamente efectiva, dado que si durante la ejecución de un nodo A del módulo de ataque se elimina a un enemigo, luego se transita a un nodo B del mismo módulo y llega en ese momento la notificación de muerte, sería la transición que conduce al nodo B la que recibiese el premio. De todos modos, con esta solución los refuerzos serán aplicados de forma coherente, ya que sólo se premiará la eliminación de un enemigo a transiciones que conduzcan a nodos de carácter ofensivo.

Sin embargo, no existe ninguna forma de mejorar el mecanismo de refuerzo negativo, puesto que al ser eliminado el propio agente, éste podía no encontrarse en un módulo concreto (por ejemplo, si muere en un ataque frontal, cabe pensar que estaría repeliendo los disparos; pero si muere fruto de un ataque por la espalda, lo lógico sería que no estuviese disparando, por lo que el módulo en ese caso sería otro distinto). Por tanto, en este caso se asignaría el refuerzo a la última transición recorrida, sea cual fuere el módulo del nodo al que conduce.

Aportación del sistema experto

Ya hemos comprobado que, de la técnica de aprendizaje por refuerzo, se conservan los mecanismos para premiar y castigar al agente ante determinados eventos del juego, así como el uso de una memoria almacenable en fichero.

En relación al sistema experto utilizado en la primera fase del ciclo de vida, en este desarrollo se reutilizará el mecanismo de sentencias condicionales para limitar el campo de acción de la funcionalidad del agente, lo cual aportará una mayor coherencia a sus actos, al tiempo que se reduce el coste computacional, ya que en aquellos ciclos que no se cumpla con las condiciones de las sentencias, no será necesario llevar a cabo ninguna modificación del comportamiento del bot.

El método *doLogic()* al que, como en el resto de prototipos, se llama de forma reiterativa durante toda la ejecución del agente, comprobará si se cumple alguna de las condiciones de una lista. En caso afirmativo, se procederá a evaluar los niveles de los sensores que conforman el estado el agente y se estudiará realizar una transición en la máquina de estados para adaptarse a la nueva situación realizando una acción distinta. Tanto si esto fuera necesario, como si no, posteriormente se ejecutará la acción elegida. Las condiciones evaluadas son las siguientes:

- Presencia de un enemigo. La aparición de un enemigo en el campo visual del agente, implica la necesidad de reaccionar antes que él.

- Ataque contra el propio agente. La percepción de que el agente está siendo dañado denota que, aunque no se vea al enemigo, éste se encuentra en algún lugar próximo (fuera del campo de visión) lanzando un ataque contra el agente y éste debe reaccionar de inmediato.

- Nodo actual específico. Agacharse, erguirse y saltar son acciones que el agente debe realizar de manera puntual, no constante. Por ello, si se comprueba que la acción anteriormente realizada fue una de las tres citadas con anterioridad, es preciso volver a sopesar la situación sensorial del agente y cambiar a otra acción para reaccionar en consecuencia.

- Ataque infundado. Si el bot se encuentra realizando una acción correspondiente a un nodo del módulo de ataque pero no ve ningún enemigo en su campo de visión, será preciso cambiar a otro nodo, ya que no resulta efectivo gastar munición de manera innecesaria.

- Solicitud de renovación de acción. Al tener como acción actual alguna relacionada con la adquisición de ítems del escenario, es preciso saber en qué momento se han adquirido para poder transitar a nodos más convenientes. Para ello, es preciso solicitar una renovación del mismo una vez que el agente reciba una notificación relativa a la adquisición de ítems.

Si fuera preciso evaluar una nueva transición, este proceso se realizaría de la siguiente manera:

1. Analizando el valor de los sensores que conforman el estado del agente (en la *Figura 17*, representado por los puntos del 1 al 8).

2. Procesando los valores de la forma más adecuada. Para ello es preciso:

   - Comprobar que el nodo actual cuenta con una transición correspondiente al estado actual, en cada uno de los enlaces que parten de él. Si no fuera así, será necesario crear dicha transición en cada uno de los enlaces, para permitir la posterior selección de alguno de ellos al cambiar de nodo. Ver puntos 9.2 y siguientes de la *Figura 17.*

   - Elegir la transición más conveniente de todas las posibles. Será necesario seleccionar una de las transiciones salientes del nodo, cuya configuración de estado coincide con la actual del agente (punto 9.3 de la *Figura 17*). Al haber una coincidencia en cada enlace, habrá que elegir aplicando los siguientes criterios:

     1. Tienen prioridad máxima aquellas transiciones que no han sido nunca utilizadas. Se persigue así la búsqueda de la transición óptima.

     2. Si todas han sido visitadas, se buscarán todas las transiciones que, a partir de un número considerable de visitas realizadas, no hayan acarreado un castigo en más de la mitad de las ocasiones.

     3. Si aún así no fuera posible encontrar una transición candidata, se elegirá aquella que, pese a tener un mal resultado, proporcione el mejor de todos los candidatos. De esta manera se elige el camino menos malo para salir de la situación actual.

   - Una vez obtenida la mejor alternativa para abandonar el nodo actual, será preciso llevar a cabo una actualización de todos los valores referenciados en la clase *Automaton* por algún motivo concreto. Es el caso de los nodos y módulos actuales, anteriores e iniciales. En determinadas situaciones de la ejecución, es necesario contar con información actualizada relativa a estos elementos. Por ejemplo, al empezar la ejecución, es necesario saber cuál será el nodo en el que se empieza (en este caso será el estado en que se recorre el escenario, puesto que proporciona un comienzo neutral al jugador). Por otro lado, es necesario saber el nodo del que se procede, puesto que al premiar o castigar una determinada conducta, en esta arquitectura se premia a la transición concreta que condujo al agente a dicha situación.

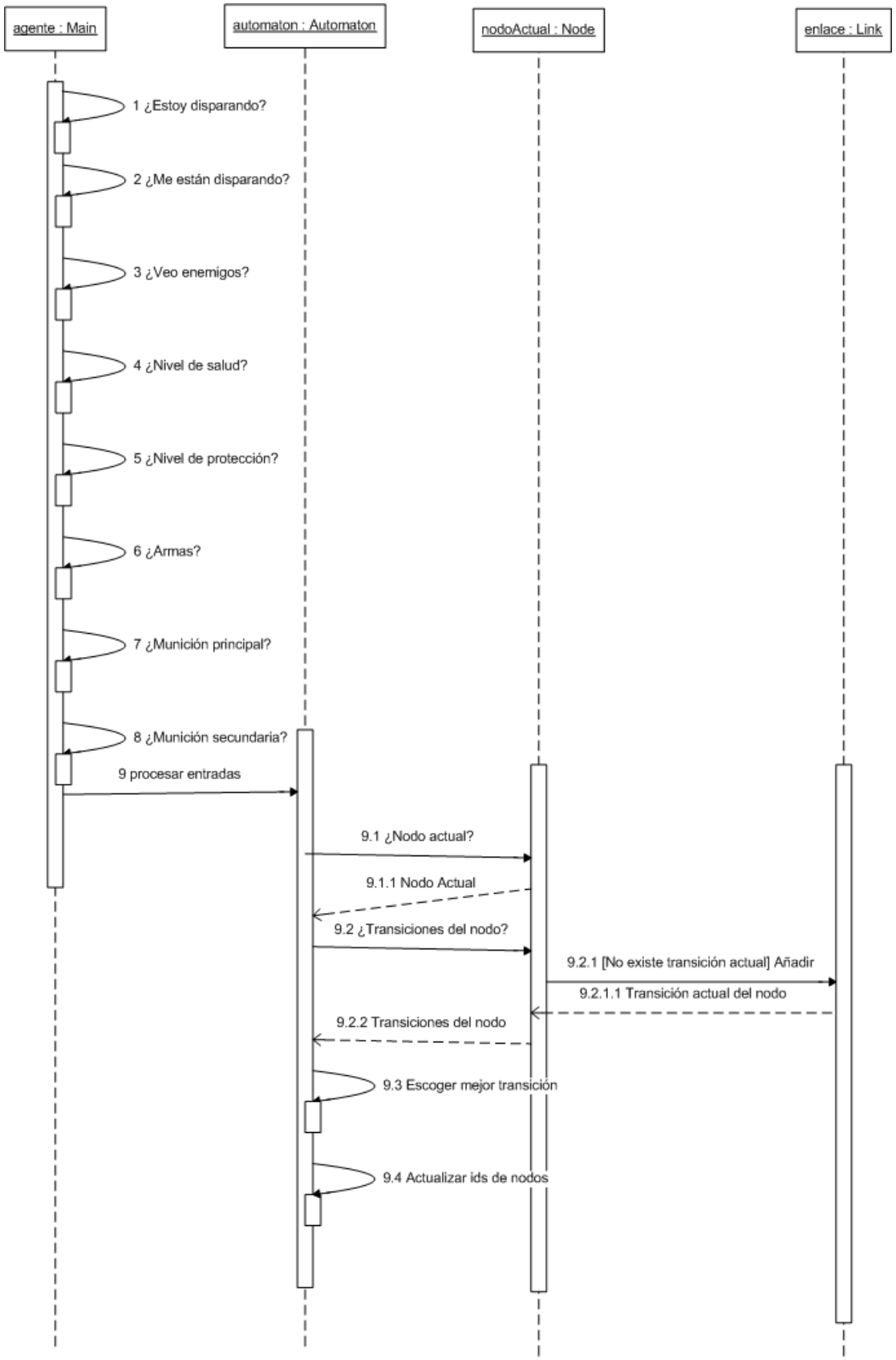

Figura 17 Diagrama de secuencia del prototipo híbrido básico

Todo el proceso anteriormente descrito, se puede comprender más fácilmente observando la *Figura 17*.

Gestión de fichero de memoria

A la hora de guardar información y restaurarla posteriormente, se ha establecido un formato jerárquico de representación de datos como el siguiente:

```
• Nodo inicial del autómata
iNode: <nombre del nodo>
• Para cada módulo del sistema
Module: <nombre del módulo>
      • Para cada nodo del módulo
      |-Node: <nombre del nodo>
            • Para cada enlace del nodo
            | |-Link: <nombre del enlace>
                  • Para cada transición del enlace
                  | | |-Transition: <nombre de la transición> Flags:
                  <niveles de los sensores> Visits: <visitas> Value:
                  <refuerzo>
            "___"
      "__"
"_"
```

Las indentaciones del texto citado, no implican una verdadera inclusión de tabuladores, sino que es la representación de estructuras repetitivas englobadas por otras de nivel superior.

Con esta metodología utilizada para procesar los datos correspondientes al comportamiento del bot, se obtiene un fichero de tamaño variable, que aumentará a medida que se definan un mayor número de transiciones por los distintos enlaces que conectan a todos los nodos entre sí, por lo que se puede hallar una relación directamente proporcional entre el volumen de información del fichero y la capacidad cognitiva del agente que lo utiliza.

Evaluación del Prototipo

Para llevar a cabo la evaluación de este prototipo, será conveniente hacerle combatir contra los dos prototipos que inspiraron su diseño. Es decir, se desarrollará una batalla contra el bot con Q Learning mejorado y otra contra el agente basado en sistemas expertos. De este modo podremos sacar conclusiones sobre las mejoras adquiridas y si es preciso evolucionar nuevamente el prototipo. Cabe destacar que, al igual que en pruebas anteriores, ésta se lleva a cabo sin un proceso previo de entrenamiento, por lo que fundamentalmente se valorará la capacidad inicial de adaptación al juego, y no tanto la evolución del sistema con el paso de las partidas, ya que dicho aspecto será evaluado en detalle en la sección de *Diseño de Experimentos y Resultados*.

A priori, se espera que el resultado de ambos combates resulte ajustado, dado que el agente basado en Q Learning tiene mecanismos similares de refuerzo a los del bot híbrido basado en máquinas de estado. Por otro lado, el nivel del agente basado en sistemas expertos resultó ser altamente competitivo, por lo que el resultado de este combate dependerá en gran medida de la capacidad del agente híbrido para hacerle frente. Los resultados de los combates son los siguientes:

Tiempo Transcurrido:	2 horas
Marcador Agente Híbrido:	1
Marcador Agente Q Learning:	0
Tasa de Igualdad:	1

Tabla 13 Resultado del combate agente híbrido vs. agente Q Learning

Como se puede observar en los datos de la tabla superior, el combate llevado a cabo mediante el agente híbrido y el bot basado en aprendizaje por refuerzo, no cumple con las expectativas generadas, al no ser capaz de alcanzar las 25 muertes. Pese a que el nuevo agente ha ganado a su predecesor, la tasa de igualdad sigue siendo igual a uno, lo que implica que no existe mucha diferencia entre ambos. Una posible explicación a este fenómeno es que ambos bots decidieron, en un momento de la partida, quedarse parados. Esto se debe a dos aspectos cruciales:

- El agente basado en Q Learning suele incurrir en un comportamiento neutral en el que no tiene que atacar ni defender, puesto que no suele reportarle malos resultados. En este caso, dicho comportamiento se refleja en su decisión de no moverse. Como ya se ha dicho en ocasiones anteriores, esto se debe a la ausencia de entrenamiento previo de este prototipo.

- El agente híbrido tiene un problema derivado de su capacidad para perseguir ítems por el escenario y cambiar de objetivo una vez que se han conseguido, ya que al obtener un nuevo elemento, permanece a la espera de recibir un mensaje que se lo notifique para, de ese modo, solicitar un cambio de acción en el método *doLogic()*. El problema es que cuando alguno de los niveles para los que busca un nuevo ítem, se encuentra al máximo, el ítem no es "consumido" al pasar por encima, lo cual implica que no se notifique al agente que lo ha recibido. Esta situación puede suponer un problema en escenarios en los que algún tipo de ítem sólo cuente con un elemento disponible en todo el mapa. En el caso de la ejecución anterior, existe un único ítem de protección en todo el mapa y, una vez que el agente híbrido pretende adquirirlo teniendo la armadura al máximo nivel, el elemento no se gasta, la notificación no llega y el bot se queda bloqueado en la posición del ítem a la espera de poder adquirirlo. Por tanto, la única posibilidad que existe de que salga de este estado, es si alguien le ataca (pero no podemos olvidar que su rival también se encontraba bloqueado en ese momento), con lo que bajaría el nivel de protección, cogería el ítem y recibiría la notificación para poder cambiar de nodo.

Pese a que el problema de esta ejecución se encuentra acotado, será necesario modificar esta situación en el futuro mediante una evolución final del prototipo híbrido.

A continuación, se muestra el resultado del segundo combate:

Tiempo Transcurrido:	38 minutos
Marcador Agente Híbrido:	13
Marcador Agente Experto:	25
Tasa de Igualdad:	1.84

Tabla 14 Resultado del combate agente híbrido vs. agente experto

Como se puede observar en los resultados de la tabla, se ha conseguido terminar la ejecución de una partida en un tiempo inferior al máximo estipulado. Esto indica que el nivel ha sido suficiente para conseguir evitar los problemas vistos anteriormente. En concreto, el agente híbrido no ha tenido problemas de bloqueo porque la acción ha sido más dinámica, lo cual ha permitido que no alcance nunca un nivel máximo en ninguno de sus sensores.

El único problema es el resultado. Ya que el bot híbrido heredaba la capacidad condicional del agente experto, se esperaba que el resultado fuese algo más ajustado. Aún así, esto tiene una explicación sencilla, ya que mientras que el agente experto es capaz de combinar diferentes acciones para realizar una estrategia más compleja (por ejemplo, disparar agachado o huir mientras salta), el bot híbrido presenta el mismo problema que en su día planteó el prototipo número dos, basado en Q Learning: al tener una única estructura que gestione el uso de todas las acciones, no es capaz de realizar varias de ellas de forma coordinada.

Este inconveniente, unido al de los bloqueos inesperados, obliga a tener que realizar una nueva etapa del ciclo de vida.

Ciclo 5: Evolución del Prototipo del Ciclo 4

Este último prototipo del ciclo de vida, se plantea como una solución a diversas carencias planteadas por su predecesor híbrido. Dicho agente planteaba problemas a la hora de cambiar el tipo de acción a realizar si la anterior no era satisfecha de forma estricta (por ejemplo, si un nivel de sus sensores se encontraba al máximo e intentaba adquirir nuevos ítems de ese tipo, quedaba bloqueado a la espera de poder consumirlos). Del mismo modo, el comportamiento de este bot, impedía realizar varias acciones de manera simultánea (a excepción de agacharse, erguirse y saltar, que presentaban excepciones explícitas dentro de las sentencias condicionales).

Por ello, además de los casos de uso contemplados en el prototipo anterior (ver *Ciclo 4: Desarrollo mediante Sistema Híbrido*), se define un caso de uso adicional que ayudará a vislumbrar una solución para parte del problema:

Figura 18 Diagrama de casos de uso del agente híbrido mejorado

Nombre	Detectar Movimiento
Descripción	Comprueba si el propio agente se encuentra parado, para evitar problemas de estancamiento de momentos específicos de la partida
Flujo Básico	El bot comprueba si está moviéndose
Postcondiciones	El bot recibe información acerca del estado de su desplazamiento espacial

CU 23 Mecanismo antibloqueo

Análisis de Requisitos

En este caso, la especificación de requisitos se mantendrá igual que en la etapa anterior del prototipo híbrido (ver *Análisis de Requisitos*), puesto que los cambios aplicados corresponden a un nivel detallado del diseño y no repercuten de manera explícita en las especificaciones del modelo.

Riesgos, Dificultades y Oportunidades de la Arquitectura

La solución que se plantea en esta etapa, pretende superar los problemas debidos a bloqueos del bot en espera de determinados eventos. Del mismo modo, existe la oportunidad de salvar un obstáculo relacionado con la imposibilidad del bot de llevar a cabo una serie de acciones de forma coordinada.

La alternativa propuesta en este caso será, al igual que ya se hizo en el prototipo tres (encargado de mejorar el bot basado en Q Learning), aportar tres estructuras distintas para gestionar acciones complementarias. Por lo que se pasará de un modelo con un autómata compuesto por todas las acciones representadas como nodos, a tener tres autómatas especializados en distintos aspectos del juego.

No obstante, existe el riesgo de que un aumento en la complejidad de la infraestructura, no conlleve una mejora de rendimiento, sino todo lo contrario (como ya sucedió al intentar mejorar el agente basado en aprendizaje por refuerzo).

Implementación de Prototipo

Como ya se ha comentado, el objetivo de este desarrollo será establecer tres autómatas especializados en diversos aspectos del comportamiento del bot. En concreto, cada uno se centrará en una de las siguientes facetas:

1. Desplazamiento horizontal. Este aspecto engloba todas las acciones relacionadas con recorrer el escenario (correr, dejar de correr o perseguir ítems por el mapa).

2. Desplazamiento vertical. Esta funcionalidad hace referencia a las acciones tomadas por el agente que afectan a su verticalidad (agacharse, erguirse o saltar).

3. Ataque. Finalmente, este autómata se encargará de la gestión de acciones ofensivas (disparar función primaria, disparar función secundaria y detener el disparo).

Dado que la estructura de clases es idéntica a la del anterior prototipo (ver *Figura 16*), en esta ocasión se mostrará un diagrama de objetos que ayude a comprender la aplicación del modelo en tiempo de ejecución:

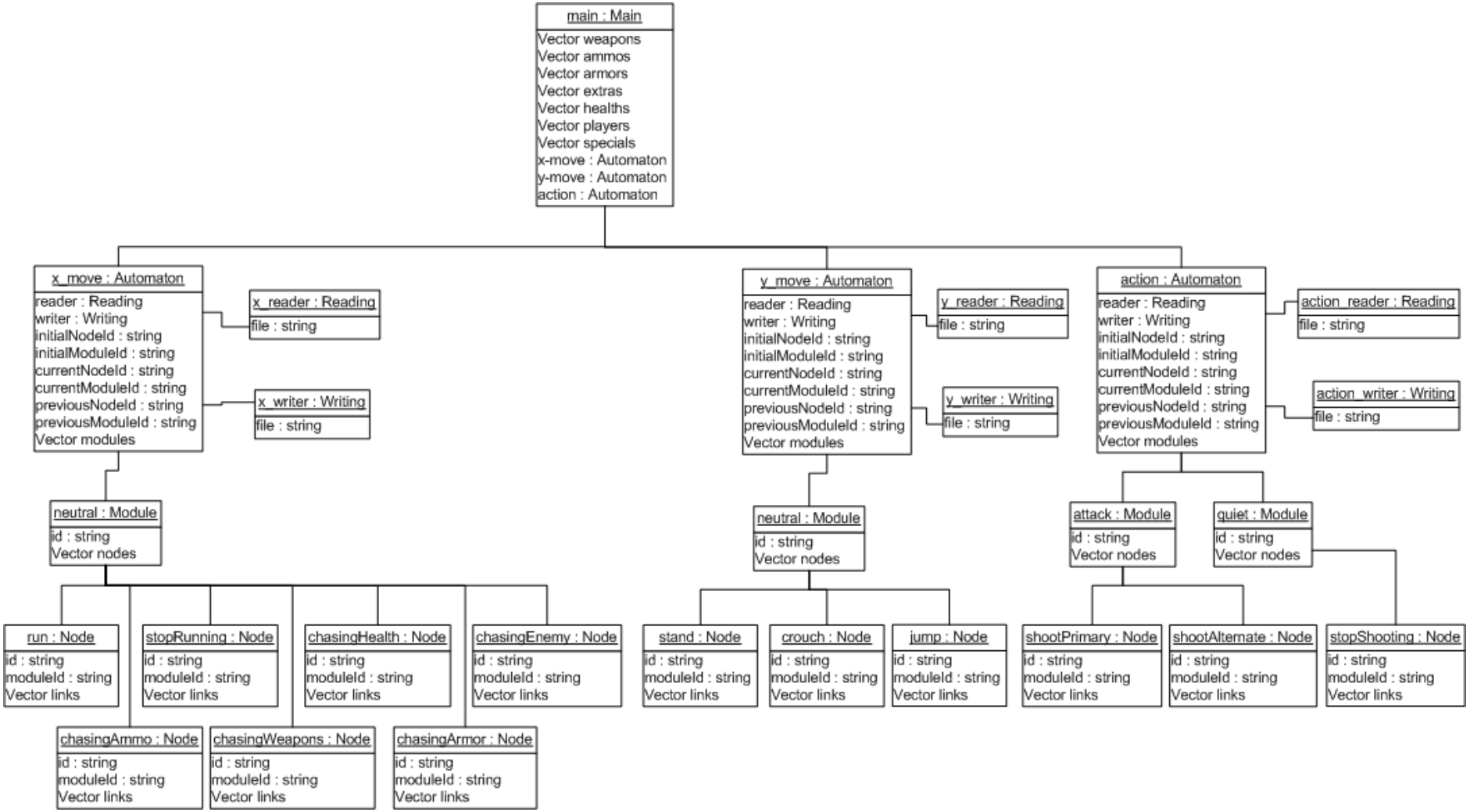

Figura 19 Diagrama de objetos del agente híbrido mejorado

Como se puede comprobar en la *Figura 19*, sigue existiendo una única clase *Automaton* de la que derivan tres objetos distintos, en lugar de una jerarquía de clases como la que se generó en el caso del modelo Q Learning mejorado. De este modo, sólo es necesario mantener varias referencias de una misma clase, las cuales contendrán distintos conjuntos de acciones sin necesidad de aplicar herencia para obtener un trato distintivo entre todas ellas. Este aspecto hace aún más adaptable a este modelo. Además de las instancias mostradas en el diagrama, para representar plenamente la situación inicial de ejecución del agente, sería necesario establecer un objeto de tipo Link por cada par de nodos pertenecientes a un mismo autómata. A medida que se va desarrollando la actividad del agente y, como ya se explicó en la *Figura 17*, se irán generando nuevas transiciones que deberán ser vinculadas a las instancias de tipo Link existentes en los distintos autómatas del sistema.

Finalmente es preciso hablar de la gestión de información en fichero. Del mismo modo que el modelo híbrido básico, este prototipo híbrido mejorado gestiona un fichero de texto en el que guarda y del que posteriormente restaura la información adquirida con el paso del tiempo durante su participación en partidas. De este modo se evita perder la experiencia adquirida en batallas previas. Para dar cabida a las nuevas mejoras estructurales implementadas, será preciso mejorar el anterior formato de almacenamiento de información, que queda de la siguiente manera:

```
• Para cada autómata del sistema
Automaton: <nombre del autómata>
• Nodo inicial del autómata
iNode: <nombre del nodo>
      • Para cada módulo del autómata
      Module: <nombre del módulo>
            • Para cada nodo del módulo
            |-Node: <nombre del nodo>
                  • Para cada enlace del nodo
                  | |-Link: <nombre del enlace>
                        • Para cada transición del enlace
                        | | |-Transition: <nombre de la transición>
                        Flags: <niveles de los sensores> Visits:
                        <visitas> Value: <refuerzo>
                  "___"
            "__"
      "_"
```

De esta manera, cuando cada autómata reciba la orden de volcar su información al fichero, estas operaciones se realizan de manera independiente y secuencial, dejando constancia de qué autómata contiene qué módulos/acciones, con lo que se consigue restaurar la información de dicho fichero de una forma fácil e inequívoca.

Evaluación del Prototipo

Como en todos los casos anteriores en los que se realiza una mejora de un modelo previo, en esta ocasión se llevará a cabo una evaluación comparativa del rendimiento de las dos versiones de agentes híbridos.

La hipótesis de esta evaluación es que, al llevar a cabo diversas mejoras de comportamiento con respecto a su predecesor, el agente híbrido mejorado derrotará al de tipo básico por una cuantiosa diferencia. A continuación, se muestran los resultados obtenidos tras una ejecución en la que ninguno de los competidores ha disfrutado de una fase de entrenamiento previa:

Tiempo Transcurrido:	1 hora y 47 minutos
Marcador Agente Híbrido Básico:	9
Marcador Agente Híbrido Mejorado:	25
Tasa de Igualdad:	2.66

Tabla 15 Resultados del combate entre versiones del agente híbrido

Finalmente podemos concluir que los resultados obtenidos satisfacen la hipótesis planteada, ya que el agente híbrido mejorado resulta mucho más efectivo que su antecesor (muestra de ello son la capacidad del nuevo prototipo para alcanzar el máximo de muertes permitido en un tiempo inferior a 2 horas, y una tasa de igualdad próxima a 3 puntos, lo que implica una diferencia notable entre la capacidad cognitiva de ambos agentes).

5. Diseño de Experimentos y Resultados

A lo largo de la sección *Trabajo Realizado*, se han llevado a cabo diversas pruebas que servían para validar cada una de las implementaciones de prototipos propuestas. No obstante, existen pruebas interesantes que no quedan contempladas dentro de estas demostraciones y que se detallan a continuación.

Representantes de cada modelo

Una vez finalizado el proceso de desarrollo, es preciso emitir valoraciones acerca de cada una de las técnicas implementadas en los agentes. Para ello, se llevarán a cabo combates entre los máximos representantes de cada técnica. Dichos representantes serán los siguientes:

- Sistemas expertos. Dado que sólo hay un caso en el que se aplique de forma exclusiva esta técnica, el bot denominado como "Agente Experto" será su representante.

- Aprendizaje por refuerzo. Esta técnica ha sido aplicada por dos prototipos de bot distintos. El que se suponía que sería una versión mejorada del primero, resultó tener un rendimiento menor, por lo que el representante de esta clase será el bot denominado "Agente Q Learning Básico".

- Sistema híbrido. En esta técnica se combinaban conceptos de las dos anteriores (sentencias condicionales de los sistemas expertos y estrategia de refuerzo de la técnica de Q Learning) junto con el uso de autómatas finitos para la implementación de una estructura de nodos que permita conmutar entre unas acciones y otras. En este caso, de los dos prototipos implementados, el segundo sí consiguió mejorar al inicial, por lo que el bot representante de esta clase será el "Agente Híbrido Mejorado".

Se estima que el resultado del combate será muy perjudicial para el agente Q Learning, dado que su comportamiento se fundamenta en el concepto de combates "uno contra uno" y, por ejemplo, cada vez que se produce una notificación de un jugador que no sea el mismo, aplica refuerzos que probablemente en un combate a tres bandas no le correspondería aplicar (por no haber sido el autor de la acción que eliminó a otro rival). Por otro lado, se espera que el combate sea tremendamente beneficioso para el bot basado en sistemas expertos, puesto que la definición de su conducta se mantiene inalterada, haya el número de enemigos que haya (si ve a alguien, evalúa riesgos e intenta disparar).

Una vez elegidos los participantes, se lleva a cabo un combate a tres bandas con el siguiente resultado:

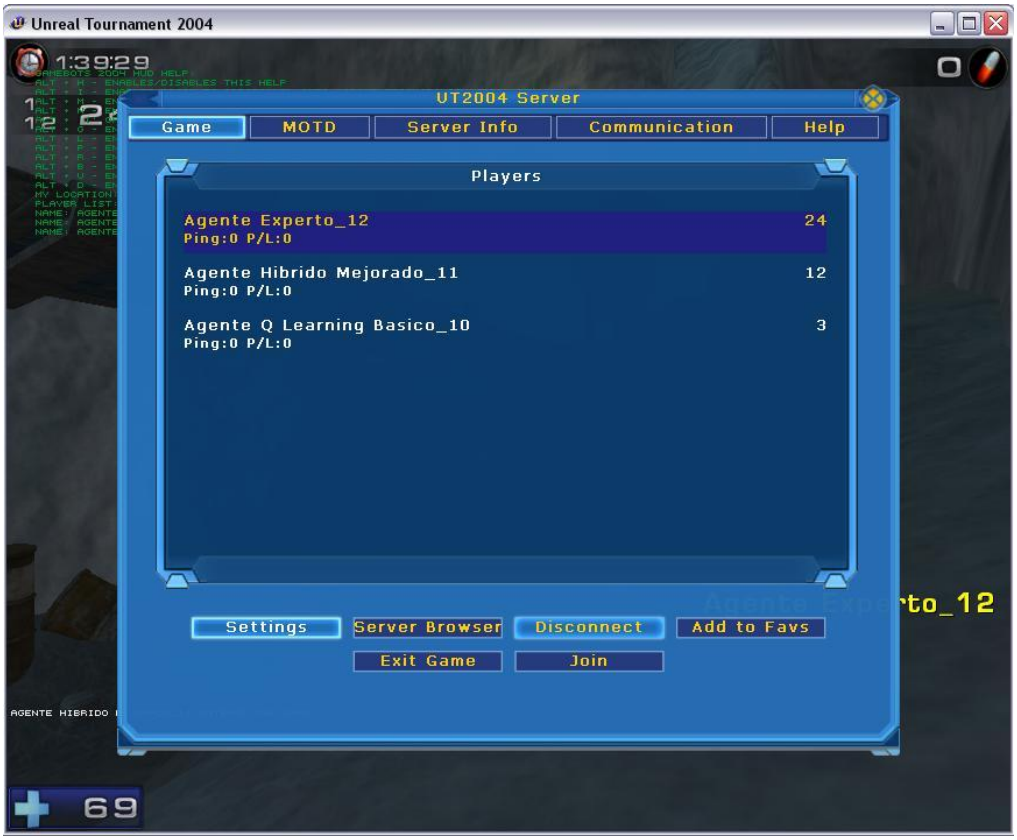

Figura 20 Resultado de combate entre representantes de cada modelo

Tiempo Transcurrido:	20 minutos y 31 segundos
Marcador Agente Experto:	25
Marcador Agente Q Learning Básico:	3
Marcador Agente Híbrido Mejorado:	12
Tasa de Igualdad entre Agentes Experto e Híbrido Mejorado:	2
Tasa de Igualdad entre Agentes Experto y Q Learning Básico:	12
Tasa de Igualdad entre Agentes Q Learning Básico e Híbrido Mejorado:	3.66

Tabla 16 Datos del combate entre representantes

Como se puede apreciar en los resultados anteriores (la *Figura 20* muestra la situación previa a que el líder de la clasificación realizase su última eliminación, puesto que después todos los jugadores son desconectados y no se podría observar los resultados), existe una gran diferencia entre el resultado obtenido por el agente experto y las otras dos implementaciones, puestos que entre las dos suman algo más de la mitad de los puntos obtenidos por la primera. No obstante, entra dentro de la situación prevista para este tipo de partida.

El mejor contra su creador

Una vez realizada la prueba anterior (ver *Representantes de cada modelo*), es el momento de incluir en el juego una modalidad hasta ahora no tenida en cuenta: el mejor agente de la prueba anterior (agente experto) luchará contra su creador (un usuario humano experto).

Pese a los buenísimos resultados obtenidos en las partidas anteriores por el agente experto, se espera que sus resultados en esta prueba sean francamente malos, puesto que su comportamiento no es más que la implementación de un subconjunto de conocimientos que posee su creador respecto a técnicas del juego. Por otro lado, el usuario humano no sólo posee más conocimientos sobre tácticas a aplicar durante el juego, sino que las que tiene en común con el agente artificial las realiza mejor que éste; sin ir más lejos, cuando el usuario humano pretende disparar a un individuo, coordina el movimiento de las piernas con el ángulo hacia el que apunta, pero el bot no es capaz de llevar a cabo acciones tan aparentemente sencillas como esa.

Una vez llevada a cabo la partida pertinente, los resultados obtenidos son los siguientes:

Tiempo Transcurrido:	9 minutos y 58 segundos
Marcador Agente Experto:	6
Marcador Usuario Experto:	25
Tasa de Igualdad:	4

Tabla 17 Combate entre agente experto y usuario experto

Como ya se vaticinó, la diferencia en el marcador final entre ambos jugadores ha resultado muy abultada. Con una tasa de igualdad de cuatro puntos y una duración de partida de diez minutos, no queda lugar a duda de que el agente artificial nunca pudo llevar la iniciativa del juego, ni tan siquiera asemejar su complejidad cognitiva a la demostrada por el usuario humano.

Adjunto a este documento, se proporciona una grabación llamada “Bot experto contra usuario experto” en la que se puede presenciar la partida, desde el punto de vista del jugador humano.

Todos contra todos

El objetivo de esta prueba, más allá de volver a evaluar la capacidad de reacción de los agentes en un escenario con múltiples rivales (como en el caso anterior), reside en estudiar la capacidad de evolución de los agentes que utilizan memoria externa para guardar información relativa a su comportamiento. Es decir, se llevarán a cabo una serie de rondas sucesivas en modo "todos contra todos" en las que los cinco prototipos pondrán a prueba sus habilidades.

La hipótesis para esta prueba es que, dado un comportamiento inicial vacío, sin entrenamiento previo de los cinco bots, podrá observarse cierta evolución de aquellos que utilizan memoria, en detrimento del agente basado en sistemas expertos, que debería ver reducida la diferencia inicial que obtenga en los resultados de las partidas iniciales ya que no utiliza un fichero para guardar información y mejorar su comportamiento.

A continuación se muestran las puntuaciones de cada agente durante los diez primeros combates llevados a cabo:

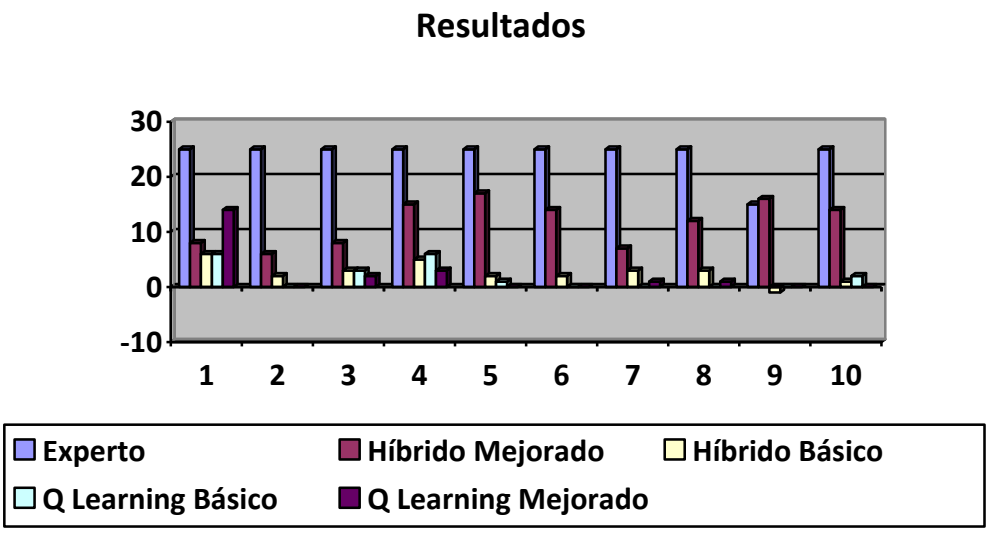

Figura 21 Evolución de partidas de "todos contra todos"

Existen numerosas lecturas respecto a los resultados obtenidos:

- Los agentes implementados mediante la técnica de Q Learning no han reportado buenos resultados. No obstante, se observa que en la primera ronda alcanzaron unos resultados bastante meritorios y, desde ese momento, su decadencia ha sido constante, rondando casi siempre los cero puntos. Este detalle redunda en la idea de que es imposible que, dada su implementación, un agente implementado con Q Learning sea capaz de generalizar correctamente, puesto que toda notificación de muerte recibida, la considera propia (con lo que se aportan refuerzos totalmente irreales que a medio plazo, resultan perjudiciales para su rendimiento).

- Los agentes híbridos han corrido suertes muy dispares. En primer lugar, el modelo básico, dadas sus limitaciones de diseño, ha tenido resultados muy mediocres a lo largo de toda la prueba (llegando a puntuar de manera negativa en las rondas finales). Por el contrario, el modelo mejorado ha conseguido resultados altamente prometedores, llegando a ser el rival más importante del líder indiscutible de la prueba (el agente experto). Este prototipo además, consiguió durante la ronda 9, el hito más importante conseguido por los bots de tipo adaptativo implementados en este proyecto, al lograr vencer al agente experto con 16 puntos, quedando todos los jugadores bloqueados en distintos puntos del escenario y consumiéndose el tiempo máximo de la partida.

- El agente experto llevado a cabo en la primera etapa del ciclo de vida del estudio, resultó ser netamente superior a todos sus sucesores. Al no utilizar memoria, no necesita adaptarse a las distintas situaciones del entorno de manera dinámica. Es decir, su comportamiento es puramente reactivo. Se ha mostrado intratable en ataque y sólo ha planteado problemas en determinados puntos del escenario en los que quedaba bloqueado por un defecto del mapa que definía un obstáculo en una trayectoria rectilínea entre dos puntos del grafo (lo cual le supuso caer derrotado en la novena ronda de juego ante el modelo híbrido mejorado).

Por tanto, como conclusión final, cabe destacar que los únicos bots que resultan apropiados para realizar partidas masivas son el agente basado en sistemas expertos y el híbrido mejorado, que combina aspectos de sistemas expertos, aprendizaje por refuerzo y autómatas finitos. Este dato vale únicamente para demostrar la adaptabilidad de ambos desarrollos a las características de la partida, puesto que en un principio se comentó que el objeto de este estudio serían las partidas de tipo *Deathmatch* (ver *Unreal Tournament 2004*).

Test de Turing

Esta última prueba del estudio tiene como objetivo llevar a cabo una aplicación práctica del test de Turing en el marco de trabajo utilizado durante todo el proyecto. Para ello, se propondrán diversos usuarios humanos con experiencia en UT04, para cumplir el rol de tribunal.

Al tribunal se le encomendará la labor de presenciar un combate en el que supuestamente juegan bots y usuarios humanos y dictaminar cuáles de los jugadores cumplen con un estilo de juego más "humano". Es decir, cuáles de ellos elaboran tácticas más complejas, reflejan emociones tales como la venganza, el miedo, etc. En realidad, la partida estará compuesta por los mismos integrantes de la prueba anterior denominada "todos contra todos" y supondrá el combate número 11 de la secuencia. Con ello se pretende que los bots sean capaces de reflejar ante el tribunal todo lo aprendido durante 10 rondas de juego.

El objetivo será por tanto, conseguir hacer creer al mayor número posible de jueces del tribunal, que el comportamiento de alguno de los agentes artificiales que participan en la partida corresponde al de un usuario humano.

En la sección *Anexo 2 – Test de Turing* se puede observar el enunciado de la prueba proporcionado a los jurados, así como las conclusiones de cada uno de ellos, organizadas en tablas. El resultado de dicho experimento es el siguiente:

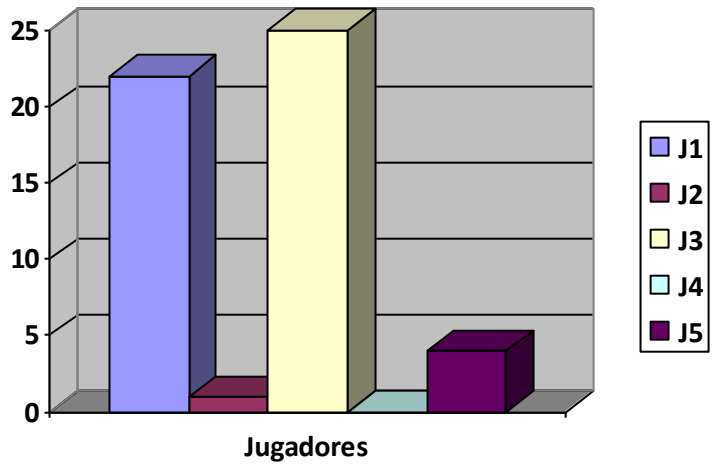

Figura 22 Marcador final del test de Turing

Los resultados observados tuvieron lugar en un margen de 20 minutos y, resulta obvio que los jugadores 1 y 3 son netamente superiores a los demás. A continuación se muestra una equivalencia que desvela la identidad real de cada jugador:

Pseudónimo	Identidad Real
Jugador 1	Agente Experto
Jugador 2	Agente Híbrido Básico
Jugador 3	Agente Híbrido Mejorado
Jugador 4	Agente Q Learning Básico
Jugador 5	Agente Q Learning Mejorado

Tabla 18 Identidad real de los participantes del test de Turing

Tras descubrir las verdaderas identidades de cada uno de los jugadores, comprobamos que el bot híbrido mejorado ha vuelto a batir al agente experto y, esta vez, además no ha sido necesario que el segundo quede bloqueado para que no resulte victorioso, ya que el agente híbrido consiguió alcanzar la cifra de 25 muertes por sus propios medios. Esto denota una clara trayectoria ascendente en el comportamiento del agente híbrido mejorado, ya que de las tres últimas rondas jugadas (rondas 9 y 10 de la prueba *Todos contra todos* y el presente test), ha conseguido ganar en dos ocasiones a su máximo rival.

A modo de resumen del contenido del apartado *Anexo 2 – Test de Turing*, las valoraciones del tribunal respecto al comportamiento de cada uno de los bots ha sido la siguiente:

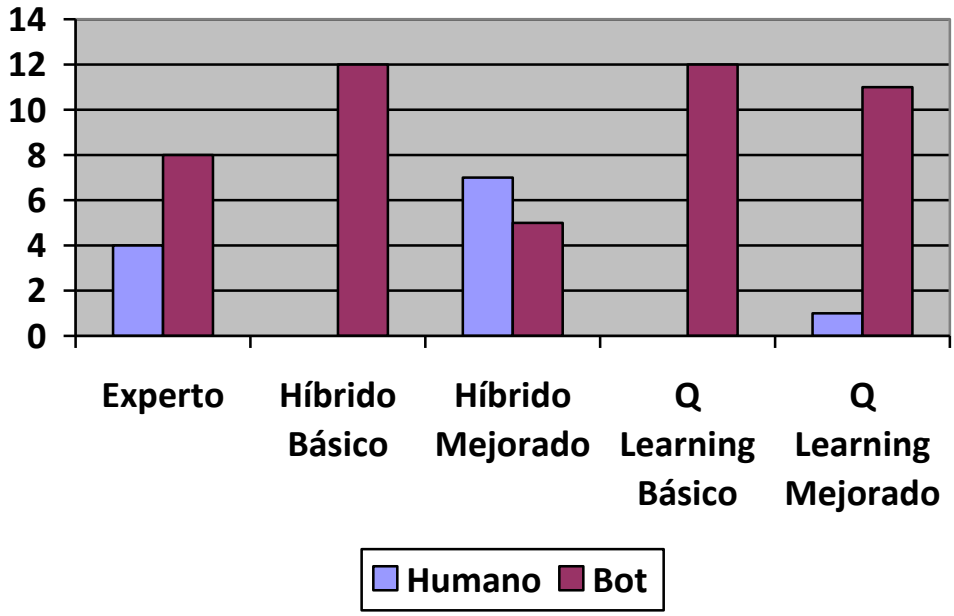

Figura 23 Valoración del tribunal del test de Turing

Como se puede comprobar, los dos únicos agentes que suscitan dudas entre los miembros del jurado a la hora de evaluar su condición humana, son los dos primeros clasificados de la partida. En el caso del agente híbrido mejorado, la controversia llega a tal punto, que un 58.33% de los miembros del jurado considera que tras él se esconde un usuario que lo maneja en tiempo real. Si se

observan las condiciones de la competición *botprize* del apartado relativo a la *Evaluación de la Capacidad Cognitiva*, se considera que un agente ha superado el test con éxito si un 80% de los miembros del jurado cree que responde a un comportamiento humano. En este caso, los resultados no llegan hasta el umbral necesario, pero son francamente positivos, ya que más de la mitad de los integrantes del comité lo consideran humano.

6. Conclusiones

A lo largo de este proyecto se ha realizado un estudio sobre la capacidad cognitiva de agentes autónomos aplicados en videojuegos. Para ello, se llevó a cabo un análisis exhaustivo de las herramientas disponibles, tanto a nivel de videojuegos, como de entornos de desarrollo que permitiesen realizar experimentos en colaboración con alguna de las anteriores plataformas de juego.

Finalmente, se optó por utilizar las siguientes herramientas:

- Plataforma de juegos. Se optó por utilizar UT04, un videojuego de tipo FPS con tintes futuristas que proporciona un mod llamado Gamebots que permite la comunicación del usuario con los agentes autónomos participantes en el juego, por medio de un protocolo de mensajes de texto.

- Lenguaje de programación. A la hora de implementar cada uno de los agentes autónomos correspondientes a los diversos prototipos de las etapas del ciclo de vida, Gamebots proporcionaba herramientas para realizarlos mediante Python y Java. Finalmente nos decantamos por la alternativa de usar Java 1.6, puesto que el desarrollo resultaba más intuitivo, gracias a la orientación a objetos.

- Entorno de desarrollo. La herramienta utilizada para llevar a cabo el proceso de implementación, compilación y depuración de código fue NetBeans, combinado con su plugin Pogamut, el cual hace posible realizar proyectos Java con la funcionalidad mínima necesaria para poder ejecutar los futuros prototipos de agente en una partida dentro de UT04.

Una vez decidido el entorno de trabajo, se llevó a cabo una serie de desarrollos sucesivos que abarcasen todo el ciclo de vida del software mediante el uso de un modelo en espiral. Todas las etapas de este modelo cumplen con las siguientes fases:

1. Análisis de requisitos. Dado que los requisitos de usuario han sido comunes a todas las etapas, en esta sección se detallaban los requisitos de usuario específicos de cada prototipo implementado.

2. Análisis de riesgos. En este apartado se estudiaba la conveniencia de implantar un modelo basado en los requisitos del apartado anterior.

3. Implementación del prototipo. En esta fase, se procedía a desarrollar el código del prototipo.

4. Evaluación del prototipo. Finalmente, se evaluaba el rendimiento de la solución desarrollada y se valoraba la posibilidad de realizar una nueva etapa del modelo en espiral con el objetivo de mejorar los resultados si estos no habían resultado convincentes.

Mediante esta metodología de trabajo, se llevaron a cabo cinco prototipos distintos, en cuyos comportamientos se aplicaron diversas técnicas de Inteligencia Artificial. En concreto:

- Prototipo 1. Se aplicaron conceptos de sistemas expertos mediante la utilización de sentencias condicionales que permitían (o no) llevar a cabo una acción si los factores evaluados relativos al propio agente y su entorno eran favorables.

- Prototipo 2. Se aplicó el algoritmo de Q Learning para reforzar el aprendizaje del agente en función del desenlace de las decisiones tomadas durante el juego.

- Prototipo 3. En este caso se llevan a cabo diversas mejoras sobre el modelo anterior que pretendían facilitar la realización de varias acciones simultáneas sin dejar por ello de utilizar mecanismos de aprendizaje por refuerzo.

- Prototipo 4. Modelo híbrido que hereda el sistema de refuerzo de acciones de los prototipos 3 y 4, y las sentencias condicionales del prototipo 1. Asimismo, se encarga de incorporar una nueva estructura al sistema: los autómatas finitos de estados.

- Prototipo 5. Finalmente se realiza una solución que pretende mejorar la anterior utilizando varios autómatas, para realizar de forma simultanea determinadas acciones durante el juego.

Tras la evaluación de todos estos prototipos se puede concluir que la capacidad cognitiva alcanzada en todos los casos, dista bastante del mínimo necesario para poder superar el test de Turing algún día. No obstante se pueden extraer distintas conclusiones dependiendo de la técnica empleada en cada caso:

- En el caso de los sistemas expertos, el agente artificial implementado resultó ser el más efectivo de todos. Pese a no alcanzar el nivel adaptativo de la escala ConsScale, posee la gran ventaja de gozar de un comportamiento constante, pues no necesita almacenar información en un fichero que le permita evolucionar con el paso de las rondas.

- Respecto a los prototipos basados en el algoritmo de Q Learning, cabe destacar que son los que peor resultado han reportado, puesto que en muchos casos se quedan quietos durante gran parte de la partida para evitar confrontaciones con los rivales. A este respecto cabe matizar que el rendimiento podría mejorar ostensiblemente con una redefinición de las situaciones objeto de refuerzo, y con un control más depurado de la recepción asíncrona de mensajes.

- Finalmente, los modelos híbridos que mezclan autómatas finitos con mecanismos de refuerzo y sistemas expertos, resultaron con toda seguridad, la alternativa más sorprendente. Responden a un paradigma adaptativo dentro de la escala ConsScale y son los únicos que han conseguido, tras una serie de rondas previas de

entrenamiento, vencer al agente basado en sistemas expertos. En concreto, la versión mejorada del código es, tras el agente experto, el prototipo más efectivo de todos los implementados. Podría verse ampliamente beneficiado de una redefinición de estados a tener en cuenta, de forma que se valoren aspectos del juego como la invulnerabilidad (propia o de los rivales) o el tipo de arma utilizada a la hora de decantarse por una acción determinada.

Por todo ello puede concluirse que la aplicación de técnicas de Inteligencia Artificial a agentes autónomos resulta tremendamente interesante de cara a analizar conductas adoptadas por los bots de cara a alcanzar sus objetivos. En concreto, el entorno de trabajo que proporciona una plataforma de videojuegos o cualquier simulador virtual, puede resultar de gran utilidad en el mundo de la investigación de técnicas y algoritmos revolucionarios, ya que se evitan los problemas derivados del uso de arquitecturas hardware, que aportan un mayor grado de complejidad e imprecisión a cualquier sistema.

En el desarrollo de este tipo de estudios, que permiten investigar aplicaciones de la IA sin apenas restricciones, reside la posibilidad de que algún día, el ser humano y los agentes que éste desarrolle lleguen a alcanzar alguno de los grados de cognición de la escala ConsScale con los que a día de hoy sólo cabe soñar.

7. Líneas Futuras

Como se deduce del apartado de *Conclusiones* de este proyecto, existe la posibilidad de continuar ampliando algunos aspectos del estudio, así como mejorando alguno de los ya existentes. Entre ellos se encuentran los siguientes:

- Aplicar técnicas más eficientes de discretización de las variables aplicadas para obtener el estado de los agentes adaptativos. De este modo, se mejoraría el tiempo de respuesta del agente, al tiempo que descendería el espacio de memoria requerido para la ejecución de cada uno de ellos. Podría plantearse la posibilidad de dotar al sistema de un mecanismo de clustering (ver *Glosario*) que simplifique la cantidad de estados a utilizar, pudiendo pasar de un conjunto de más de 1000 estados, a uno de tan solo varias decenas.

- Tener en cuenta aspectos más complejos gestionados por Gamebots. Algunos de los defectos observados en los prototipos durante las diversas fases de pruebas, denotan que mediante el tratamiento de otros aspectos del juego, los resultados habrían podido ser mucho mejores. Sin ir más lejos, los siguientes aspectos podrían ser aplicados en futuros desarrollos:

    - Consciencia del arma utilizada. Tener en cuenta que tipo de arma se utiliza en cada momento y las repercusiones que implique dentro del juego, resultan aspectos muy interesantes de cara a optimizar las estrategias de ataque de los agentes en desarrollos futuros.

    - Evaluación de invulnerabilidad de los jugadores (tanto del propio agente, como de sus rivales). No es necesario, ni recomendable atacar a alguien que es inmune a las balas. Del mismo modo, no es necesario adoptar estrategias defensivas en condiciones de invulnerabilidad del propio agente.

    - Necesidad de recoger determinados ítems. Tampoco es aconsejable invertir tiempo en recoger ítems de manera periódica, si estos no son necesarios en un momento puntual (por ejemplo, no hace falta buscar ítems de salud si el nivel del agente es máximo).

    - Persecución de rastros. Como evolución de la funcionalidad que poseen algunos de los prototipos, sería aconsejable desarrollar una funcionalidad que permita al agente ir tras un enemigo, incluso si éste es perdido de vista al esconder tras un obstáculo o doblar una esquina.

- Probar otras técnicas de IA, de cara a poder valorar la repercusión que podrían alcanzar en el comportamiento de los futuros prototipos. Entre otras, podrían realizarse implementaciones con técnicas como:

  - Redes de neuronas artificiales, puesto que es una técnica tolerante a fallos y se ha probado con éxito en arquitecturas similares (NeuralBot [*33*], uno de los bots diseñados para Quake II basaba sus decisiones en las salidas de una red de neuronas).

  - Lógica difusa, ya que la definición de estados llevada a cabo en el modelo híbrido, en la que se discretizaron los valores para acabar teniendo niveles orientativos, resultan una base muy interesante para este tipo de técnica.

- Aplicar técnicas más efectivas de sincronización de eventos. Uno de los principales problemas a los que se ha hecho frente durante el desarrollo de los modelos adaptativos (prototipos con aprendizaje por refuerzo y autómatas finitos) fue el de poder determinar el estado y acción concretos que realizaba el bot en el momento en que se produjo un evento que posteriormente le era enviado sin información relativa al dicho instante de tiempo. La incorrecta asignación de refuerzos a estados del sistema implica una repercusión negativa en el desarrollo del agente, puesto que se fomentan situaciones no positivas o, en el peor de los casos, contraproducentes.

8. Bibliografía

- **Información sobre la arquitectura de Pogamut**
    - Web de Pogamut:

      http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php

- **Información sobre la arquitectura de Gamebots**
    - Web de Gamebots:

      http://Gamebots.planetunreal.gamespy.com/
    - Web de Pogamut:

      http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php

- **Categorías de videojuegos**
    - Artículo de Wikipedia sobre el mundo de los videojuegos:

      http://es.wikipedia.org/wiki/Desarrollador_de_Juegos_de_Video

- **Sistemas expertos**
    - Sistemas Expertos – Aplicaciones de la inteligencia artificial en la actividad empresarial (1988). Paul Harmon y David King. Ed. Díaz de Santos:

      http://books.google.es/books?id=QZ3C7-y6LxAC
    - Definición de Wikipedia de sistema experto:

      http://es.wikipedia.org/wiki/Sistema_experto

- **Aprendizaje por refuerzo**
    - Reinforcement Learning: An Introduction (2005). Richard S. Sutton y Andrew G. Barto. MIT Press:

      http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html
    - Código de apoyo de aprendizaje por refuerzo en videojuegos:

      http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

- Aplicación práctica de aprendizaje por refuerzo en videojuegos:

  http://courses.ece.ubc.ca/592/PDFfiles/Reinforcement_Learning2_c.pdf

- Algoritmo *Q Learning*:

  http://en.wikipedia.org/wiki/Q-learning

- Algoritmo *SARSA*:

  http://en.wikipedia.org/wiki/SARSA

- Algoritmo *Temporal Difference*:

  http://en.wikipedia.org/wiki/Temporal_difference_learning

9. Referencias

1. **Cambridge Advanced Learner's Dictionary:**

   http://dictionary.cambridge.org/

2. **Diccionario de la Real Academia Española de la Lengua:**

   http://www.rae.es/rae.html

3. **The Entertainment Software Association**

   http://www.theesa.com/

4. **Microsoft**

   http://www.microsoft.com/

5. **Sony**

   http://www.sony.com/

6. **Nintendo**

   http://www.nintendo.es/

7. **ConsScale**

   http://www.conscious-robots.com/consscale/

8. **Página de descargas de la familia de estándares PSS**

   http://www.esa.int/TEC/Software_engineering_and_standardisation/TECBUCUXBQE_2.html

9. **Agencia Espacial Europea**

   http://www.esa.int/esaCP/Spain.html

10. **BotPrize**

    http://botprize.org/

11. **Saga Call of Duty**

    http://www.callofduty.com/

12. **Saga Ghost Recon**

    http://www.ghostrecon.com/

13. **Saga Half Life**

    http://half-life2.com/

14. Saga Counter Strike

http://planethalflife.gamespy.com/cs/

15. Saga Unreal Tournament

http://planetunreal.gamespy.com/View.php?view=UTGameInfo.Detail&id=1

16. Saga Quake

http://planetquake.gamespy.com/

17. Test Minimental

http://es.wikipedia.org/wiki/Mini-mental_state_examination

18. Herzog Zwei

http://en.wikipedia.org/wiki/Herzog_Zwei

19. Dune II

http://duneii.com/

20. Battlecruiser 3000AD

http://www.3000ad.com/

21. Golden Eye

http://en.wikipedia.org/wiki/GoldenEye_007

22. Halo

http://www.microsoft.com/Games/Halo/

23. FarCry

http://www.farcry2.com/

24. F. E. A. R.

http://www.whatisfear.com/

25. The Elder Scrolls IV: Oblivion

http://www.elderscrolls.com/home/home.php

26. Licencia GPL

http://www.gnu.org/licenses/licenses.es.html

27. Especificaciones de QuakeC 1.0

http://www.gamers.org/dEngine/quake/spec/quake-spec34/qc-menu.htm

28. Pogamut

http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php

29. Gamebots

http://Gamebots.planetunreal.gamespy.com/

30. NetBeans

http://www.netbeans.org/

31. Lenguaje de programación Java

http://java.sun.com/

32. Lenguaje de programación Python

http://www.python.org/

33. Neuralbot

http://homepages.paradise.net.nz/nickamy/neuralbot/index.html

Glosario

- **Usuario.** Se considerará un usuario del juego a la persona física que utilice el producto.
- ***Shooter*.** Nombre utilizado para identificar a los juegos considerados "de disparos". A lo largo de este documento se utilizará dicho anglicismo por ser un término aceptado a nivel mundial.
- **Jugador.** Se utilizará este término para designar a todos los participantes de alguna modalidad de juego, independientemente de si estos son controlados por el usuario o no.
- **2D.** Término empleado para hacer referencia a elementos bidimensionales.
- **Agente Autónomo.** Concepto que alude a aquellos jugadores capaces de tomar sus propias decisiones durante el juego. A lo largo de este documento también se les llama agentes autónomos virtuales (dado que no son agentes físicos), agentes inteligentes, personajes sintéticos, bots (diminutivo de robots), AI Bots, bots inteligentes, etc.
- **Mod.** Diminutivo del término inglés "modification". A lo largo del documento se utiliza para hacer referencia a módulos adicionales de determinados juegos, que extienden la funcionalidad original proporcionando nuevas posibilidades, ambientaciones, objetos, etc.
- **Offline.** Término inglés utilizado para referirse a entornos o sistemas sin conexión a una red de usuarios (tales como un área local o Internet).
- **Online.** Término inglés utilizado para referirse a entornos o sistemas con conexión a una red de usuarios.
- **NPC.** Siglas del término inglés "Non-Player Character". Utilizado como sinónimo de bot y agente autónomo.
- **Waypoint.** A lo largo de este documento, se tratan como la estructura de coordenadas para ubicar puntos de referencia tridimensionales en un escenario de la plataforma de juegos Counter Strike con el fin de definir el circuito que deberán recorrer los bots durante la ejecución de una partida.
- **Licencia GPL.** Licencia empleada para publicar productos de software libre, que permite a los usuarios publicar versiones modificadas de dicho producto.
- **Plugin.** Posee el mismo significado que el término "mod", pero su aplicación es de carácter más genérico. En este documento se utiliza para hablar de Pogamut como un complemento del entorno de desarrollo NetBeans.

- **ASCII.** Siglas del "American Standard Code Information Interchange", traducidas al español como código estándar estadounidense para intercambio de información. Consiste en un código de caracteres basado en el alfabeto latino tal como se usa en el inglés moderno y otras lenguas occidentales. Cada carácter es representado mediante una cadena de 8 bits.

- **API.** Del inglés "Application Programming Interface", se traduce a español como la Interfaz de Programación de Aplicaciones y está formada por el conjunto de procedimientos que ofrece un determinado componente para poder ser utilizado como una capa de abstracción.

- **Clustering.** Técnica utilizada para agrupar conjuntos de elementos que pueden desempeñar su labor de manera conjunta.

Anexos

Anexo 1 – Planificación y Presupuesto

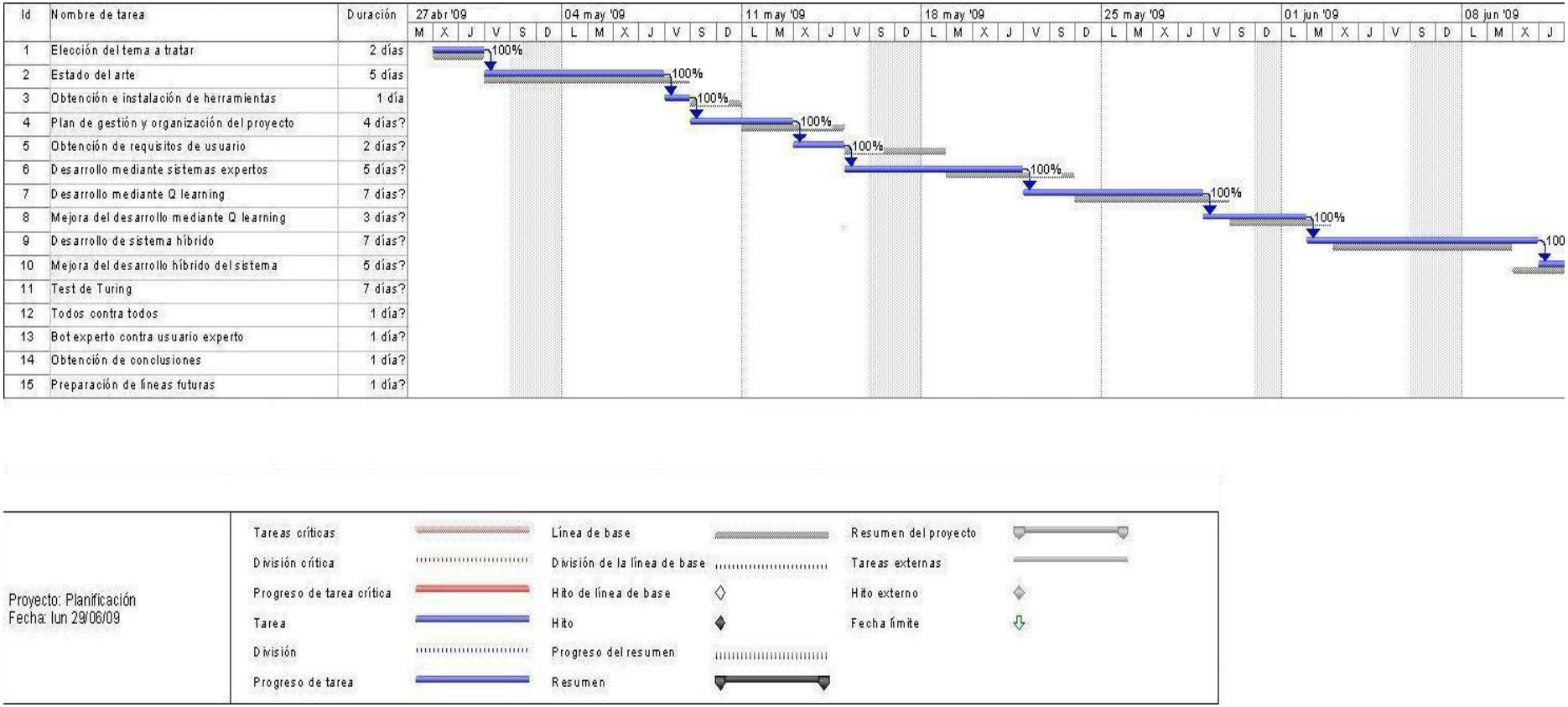

Figura 24 Planificación del proyecto (hoja 1)

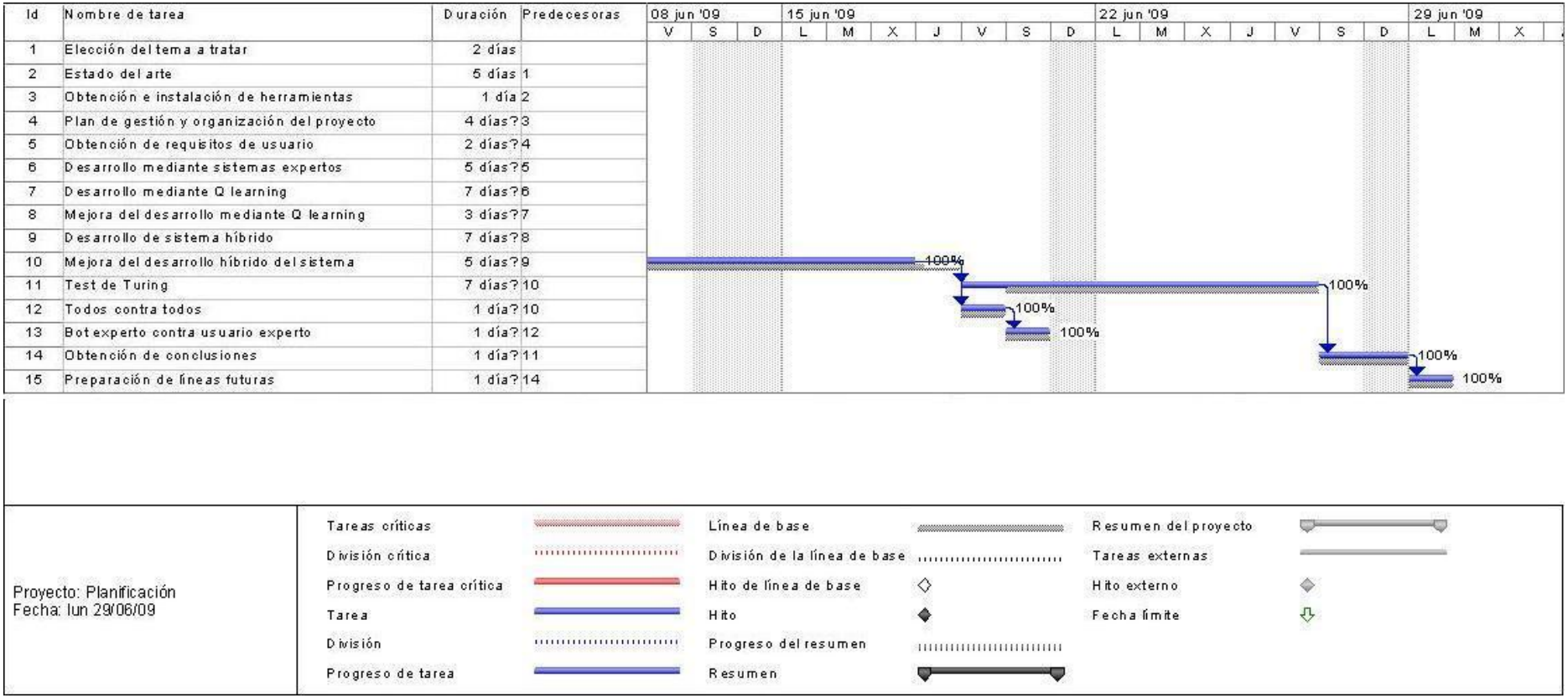

Figura 25 Planificación del proyecto (hoja 2)

Como se puede observar en las *Figura 24* y *Figura* 25, no existe ningún retraso en cuanto a la planificación general del proyecto, ya que aunque sí existen pequeñas variaciones puntuales en tareas específicas del mismo, éstas no afectan a la fecha final de entrega al compensarse entre sí. En cuanto al presupuesto, el coste final del proyecto, basado en la planificación es el siguiente:

Duración	Trabajo	Coste
52 días	436 horas	9.340 €

Tabla 19 Coste del proyecto

Se han tomado como referencia jornadas laborales de 8 horas y las siguientes tarifas de mercado para la retribución de las tareas realizadas:

Cargo Profesional	Retribución (hora hombre)	Tareas Desempeñadas
Jefe de Proyecto	30 €	1, 3, 14 y 15
Analista	25 €	2, 4, 5, 14 y 15
Programador	20 €	6, 7, 8, 9 y 10
Encargado de Pruebas	15 €	11, 12 y 13

Tabla 20 Retribuciones por cargo

Anexo 2 – Test de Turing

A continuación, se muestra el enunciado planteado a los miembros del tribunal para la realización de este test:

Esta prueba consiste en llevar a cabo el test de Turing aplicado a la plataforma de videojuegos Unreal Tournament 2004 y, más concretamente, sobre el comportamiento de los bots que participan en una partida de dicho juego. En esencia, este test consiste en que un jurado se encargue de observar el comportamiento de los jugadores y decidan si cada uno de ellos es un bot, o por el contrario, hay alguien en su ordenador manejándolo con el teclado y el ratón.

En este caso, Uds. serán el jurado de una partida para la que tienen que opinar en igualdad de condiciones (es decir, todos tendrán que ver los mismos aspectos de la partida al mismo tiempo). Como existen limitaciones técnicas debido al uso de licencias del cliente gráfico de Unreal Tournament 2004, se les proporciona un video de una partida organizada por Internet. En dicho video, la cámara va rotando por todos los jugadores cada cierto tiempo para que se pueda apreciar el comportamiento de cada uno de ellos en varios instantes del juego. Las condiciones de la partida son las siguientes:

- *La partida tiene una duración máxima de 2 horas (en realidad la que se adjunta termina en 20 minutos).*
- *La partida acaba si alguien consigue alcanzar un marcador de 25 muertes durante la partida.*
- *Existen distintos elementos repartidos por todo el escenario que aportan salud, protección, munición y armamento a los jugadores que pasan por encima de ellos.*
- *En la partida creada, hay 5 jugadores (abajo a la izquierda pone el nombre de cada uno), de los cuales un número indeterminado (mínimo, 0; máximo, 5) es controlado por humanos y el resto serán bots.*

Además del video, se adjunta una tabla en la que deberán dar su opinión acerca de cada jugador (si es humano o un bot). Además, deberían añadir una justificación en cada caso.

El video proporcionado a los miembros del jurado, se encontrará adjunto a este documento con el nombre de "test de Turing". Finalmente, los resultados obtenidos se muestran a continuación, con el mismo formato de tabla facilitado a los participantes del experimento:

	Jugador	¿Humano o Bot?	Comentarios
Jurado 1	1	Humano	Comportamiento humano lógico.
	2	Bot	No hace nada.
	3	Humano	Es vengativo.
	4	Bot	Huye de los disparos.
	5	Bot	No dispara.

Tabla 21 Evaluaciones del test de Turing del jurado 1

	Jugador	¿Humano o Bot?	Comentarios
Jurado 2	1	Bot	Al principio parece humano ya que sigue una especie de estrategia de acercamiento a los rivales, pero luego se pone a saltar adelante y atrás. No creo que este apuntando a nadie.
	2	Bot	No se mueve nada. No es una estrategia propia de un humano.
	3	Humano	Este ya tiene un comportamiento más humanizado. Se mueve buscando víctimas y dispara cuando las detecta.
	4	Bot	No se mueve.
	5	Bot	Comportamiento totalmente aleatorio.

Tabla 22 Evaluaciones del test de Turing del jurado 2

	Jugador	¿Humano o Bot?	Comentarios
Jurado 3	1	Bot	Salta mucho y no dispara hasta que está encima del contrario.
	2	Bot	No dispara al enemigo.
	3	Humano	Se esconde mucho, es eficaz (ha ganado, pero a veces actúa raro).
	4	Bot	Comportamiento extraño: NO se mueve, dispara al suelo, no dispara al enemigo.
	5	Bot	Comportamiento repetitivo, realiza un comportamiento extraño de vaivén.

Tabla 23 Evaluaciones del test de Turing del jurado 3

	Jugador	¿Humano o Bot?	Comentarios
Jurado 4	**1**	Bot	Muchos saltos impropios de un humano. Es demasiado descarado al atacar, pero no se le da mal matar.
	2	Bot	Este no se mueve.
	3	Bot	Este parece más espabilado, pero al principio persigue a un bot sin apenas disparar. Por sus movimientos, es el más cercano a un comportamiento humano. Si tuviera que elegir un humano, diría que es este, aunque me parece que tampoco lo es.
	4	Bot	O es un bot inútil o el jugador lo había dejado solo mientras seguía la partida. Al principio dispara contra todo, pero sin alcanzar al enemigo.
	5	Bot	Varios movimientos incomprensibles, pero parece que no es tan descarado atacando. Va algo más lento, se para alguna vez, lo que se puede interpretar como que el jugador está observando el panorama, dónde hay armas, pelea… aún así, para mi es un bot.

Tabla 24 Evaluaciones del test de Turing del jurado 4

	Jugador	¿Humano o Bot?	Comentarios
Jurado 5	**1**	Bot	Realiza movimientos poco humanos.
	2	Bot	Ningún humano se mueve así. Además se queda delante de otros sin atacar.
	3	Bot	Es el que tiene un comportamiento más humano, se cubre, y ataca, pero de todas formas ¿por qué salta tanto? No lo veo humano.
	4	Bot	Se mueve extraño. Además es poco convincente buscar y matar.
	5	Bot	No dispara decididamente cuando ve a otro bot, sino que dispara ráfagas sueltas. Además, el movimiento de saltar y volver atrás no es natural.

Tabla 25 Evaluaciones del test de Turing del jurado 5

	Jugador	¿Humano o Bot?	Comentarios
Jurado 6	**1**	Humano	Salta mucho, combate bastante aceptable, recolección de objetos buena, sigue movimientos en direcciones concretas.
	2	Bot	No se mueve ni dispara estando enfrente del enemigo.
	3	Bot	Muchos saltos y ratos agachado, en ocasiones parece esconderse, aceptable disparando, en ocasiones ignora al enemigo.
	4	Bot	Movimientos aparentemente aleatorios, dispara erróneamente y en ocasiones se muestra inerte.
	5	Bot	Movimientos bruscos, inmóvil o girando en ocasiones, ignora totalmente al enemigo y no dispara cuando debe.

Tabla 26 Evaluaciones del test de Turing del jurado 6

	Jugador	¿Humano o Bot?	Comentarios
Jurado 7	**1**	Humano	Por su forma de esconderse y de moverse.
	2	Bot	Porque no se mueve nada.
	3	Humano	Por su forma de moverse y apuntar a los enemigos.
	4	Bot	Porque a veces dispara a puntos donde no hay nadie.
	5	Bot	Porque hace saltos repetitivos de ida y vuelta.

Tabla 27 Evaluaciones del test de Turing del jurado 7

	Jugador	¿Humano o Bot?	Comentarios
Jurado 8	1	Humano	Lo único que veo es que en ocasiones se traba cuando no ve ningún rival. Por lo demás me parece que juega bastante bien (incluso mejor que yo).
	2	Bot	No le he visto moverse en toda la partida.
	3	Humano	Sigue a los rivales y los aniquila.
	4	Bot	No consigue apuntar correctamente a sus rivales ni se interesa por observar el comportamiento de los mismos.
	5	Bot	Se queda muy atascado cuando se cae de la plataforma e intenta volver a ella. Además, hay situaciones en las que tiene rivales cerca y no les ataca.

Tabla 28 Evaluaciones del test de Turing del jurado 8

	Jugador	¿Humano o Bot?	Comentarios
Jurado 9	1	Bot	Hace movimientos bastante raros e incomprensibles, como ponerse a saltar en una zona determinada del mapa o no seguir a sus enemigos.
	2	Bot	Pasividad completa a pesar de tener adversarios delante.
	3	Humano	Sigue a los enemigos, busca armas, utiliza el elevador correctamente y dispara a larga distancia.
	4	Bot	Mismo comportamiento que el jugador 2.
	5	Bot	A pesar de tener enemigos delante no dispara, no reacciona ante algunos ataques y tiene movimientos incomprensibles.

Tabla 29 Evaluaciones del test de Turing del jurado 9

	Jugador	¿Humano o Bot?	Comentarios
Jurado 10	**1**	Bot	Cuando tiene poca vida y pasa cerca de ítems de salud, no se molesta en cogerlos. Del mismo modo, cuando reaparece en el escenario (sin apenas armas), no establece como prioridad el recoger armamento. Finalmente, a veces se queda saltando sin sentido.
	2	Bot	No dispara a los enemigos que tiene en el punto de mira.
	3	Bot	Va siempre agachado y salta cuando le disparan, lo que resulta un comportamiento propio de un usuario humano, pero hay detalles extraños como quedarse en una esquina dando la espalda al resto del entorno.
	4	Bot	Al igual que el jugador 2, no dispara cuando tiene enemigos delante.
	5	Bot	Igual que con los jugadores 2 y 4.

Tabla 30 Evaluaciones del test de Turing del jurado 10

	Jugador	¿Humano o Bot?	Comentarios
Jurado 11	**1**	Bot	Va a por enemigos distintos sin terminar de matarlos. En el instante 1:40 del video, se pone a saltar adelante y atrás repetidamente en una zona sin armamento ni enemigos. Realiza desplazamientos sin sentido.
	2	Bot	Ante un enemigo, no dispara.
	3	Humano	Se mueve dependiendo de las circunstancias, es decir, si ve un objetivo va a por él, si ve un arma la recoge, etc. Su desplazamiento no parece ser mecánico.
	4	Bot	Como en el caso del jugador 2, no dispara cuando hay enemigos cerca.
	5	Humano	Examina zonas con miedo. No es mecánico. Se esconde e intenta disparar desde lejos. Coge armas más potentes cuando es posible.

Tabla 31 Evaluaciones del test de Turing del jurado 11

	Jugador	¿Humano o Bot?	Comentarios
Jurado 12	**1**	Bot	Decente peleando pero muy torpe moviéndose (mucho rato saltando y da cabezazos contra las paredes).
	2	Bot	Es una lacra.
	3	Bot	Similar al uno (tal vez un poco peor). Resulta decente peleando y muy torpe moviéndose (camina agachado).
	4	Bot	Ni dispara ni se mueve.
	5	Bot	Muy torpe en todo.

Tabla 32 Evaluaciones del test de Turing del jurado 12

Anexo 3 – Contenido del DVD

La estructura del DVD en el que se encuentra contenido este documento es la siguiente:

1. Directorio *Código*. Contiene los cinco proyectos ejecutables con Pogamut en el entorno de programación Netbeans, correspondientes a cada uno de los modelos implementados durante la sección de *Trabajo Realizado*. Asimismo, se encontrará un directorio llamado "Ficheros de Experiencia", que contendrá los ficheros con las memorias que cada uno de los cuatro bots adaptativos desarrollaron a lo largo de las diez partidas desarrolladas durante la prueba *Todos contra todos* anteriormente descrita. Para poder hacer uso de dichas memorias, los ficheros deberán ser ubicados (sin las carpetas en que se encuentran contenidos) en el directorio de instalación de Netbeans 6.1.

2. Directorio *Documentación*. Contiene los siguientes archivos:

    a. *Diagramas UML.vsd*. Incluye todos los diagramas UML desarrollados a lo largo del proyecto.

    b. *Test de Turing – Plantilla de Evaluación.xls*. Formato de tabla proporcionado a los miembros del jurado de la prueba *Test de Turing*.

    c. Directorio *Javadoc*. Contiene toda la documentación del código de cada uno de los proyecto Pogamut proporcionados en la carpeta de Código.

    d. *Planificación.mpp*. Contiene la planificación y presupuesto final del proyecto.

    e. *Memoria.pdf*. Contiene la memoria del proyecto en formato *pdf*.

    f. *Memoria.doc*. Contiene la memoria del proyecto en formato *doc*.

3. Directorio *Herramientas*. Contiene todas las aplicaciones indispensables para el desarrollo del proyecto. El orden en que deben ser instaladas en el sistema es el siguiente:

    a. Kit de desarrollo de Java.

    b. Netbeans 6.1.

    c. Pogamut 2.

    d. Servidor dedicado de UT04.

    e. Parche de actualización UT04.

4. Directorio *Imágenes*. Contiene todas las imágenes insertadas en la memoria del proyecto.

5. Directorio *Videos*. Contiene un video en el que se puede observar el desarrollo de la prueba llamada *El mejor contra su creador* y otro sobre el experimento del *Test de Turing*.

6. Archivo *LEEME.txt*. Contiene la información de este anexo e información relativa al autor.

Anexo 4 – Modo de Ejecución

El proceso para llevar a cabo la ejecución de un bot sintético en UT04 es el siguiente:

1. Lanzar una partida de tipo *Deathmatch* mediante el script proporcionado por Pogamut en su directorio de instalación. Esto iniciará un servidor dedicado con una duración máxima de 2 horas y un límite de muertes por agente y partida igual a 25.

2. Añadir el bot sintético a la partida. Esto puede hacerse de dos formas distintas:

    a. Ejecutando el archivo *.jar* incluido dentro del directorio *dist* del proyecto Pogamut que contiene el código del agente (mediante el uso de un intérprete de comandos o haciendo doble clic sobre el archivo).

    b. Arrancando el entorno de trabajo Netbeans 6.1. Acto seguido será preciso abrir el proyecto correspondiente al agente que se quiere ejecutar. Finalmente, se lanzará el jugador pulsando sobre el botón "*Run*" del entorno de programación. Mediante el uso de esta alternativa, la vista "*services*" del entorno permite terminar la ejecución del agente en cuestión, pulsando sobre él con el botón derecho del ratón y pulsando en la opción "*Terminate*" del menú contextual (ver *Figura 26*).

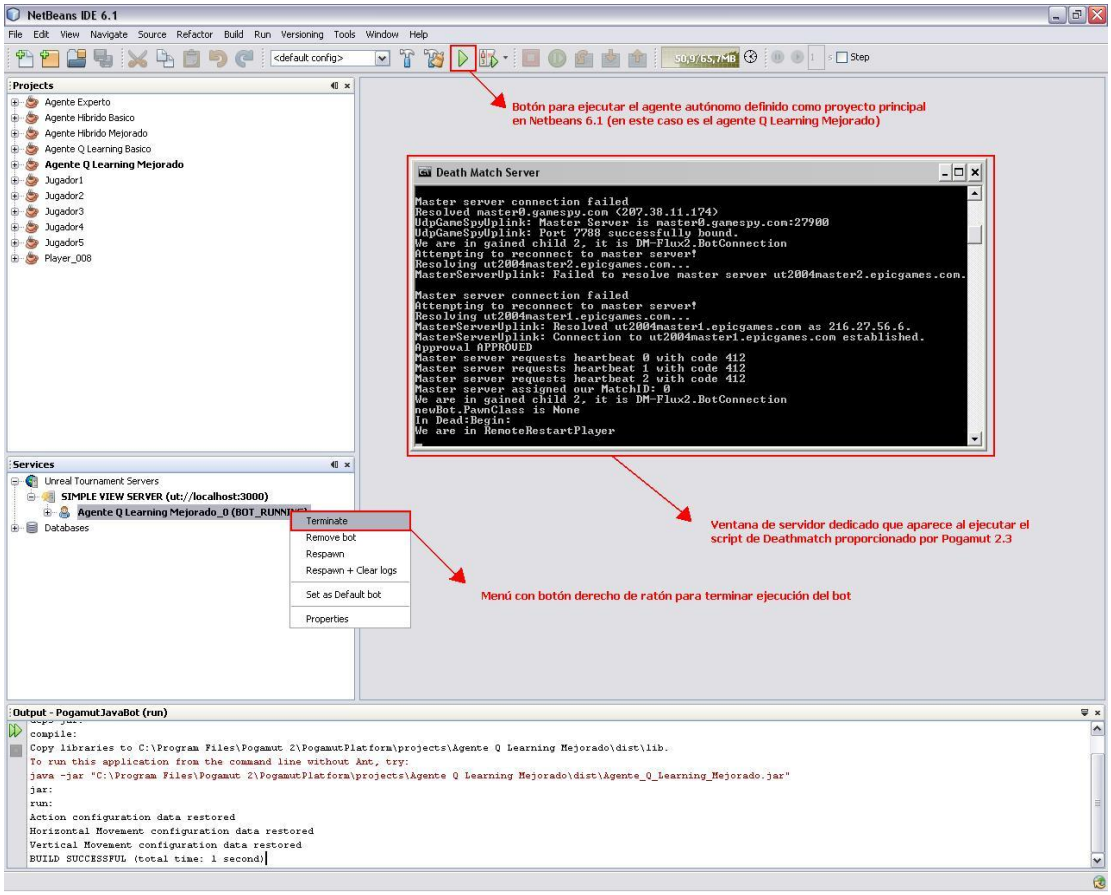

Figura 26 Ejecución de un agente autónomo